TEKNIIKKA JA LIIKENNE

Tietotekniikka

Ohjelmistotekniikka

INSINÖÖRITYÖ

Automaattisen testauksen ja jatkuvan integraatiojärjestelmän toteuttaminen

Työn tekijä: Sami Sinisalo
Työn ohjaajat: Auvo Häkkinen,
Mika Huhtamäki

Työ hyväksytty: ___. ___. 2010 **Auvo Häkkinen**

TIIVISTELMÄ

Työn tekijä: Sami Sinisalo

Työn nimi: Automaatisen testauksen ja jatkuvan integraatiojärjestelmän toteuttaminen

Päivämäärä: 21.05.2010 **Sivumäärä:** 39 s. + 7 liitettä

Koulutusohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Työn ohjaaja: Yliopettaja Auvo Häkkinen, Metropolia Ammattikorkeakoulu

Työn ohjaaja: Teknologiajohtaja (CTO) Mika Huhtamäki, Aspida Oy

Aspida Oy tuottaa ja toteuttaa Internet-pohjaisia ohjelmistoja tiedon analysointiin ja esittämiseen. Testaus kuuluu yhtenä osana kaikkiin yrityksen ohjelmistokehitysprojekteihin. Tällä tavoin pidetään yllä yrityksen ohjelmistojen korkeaa laatua ja varmistetaan ohjelmistojen toimivuus. Testauksen automatisointi vaatii paljon taustatyötä ja suunnittelua ennen lopullista toteutusta. Oikeiden komponenttien, ohjelmien ja menetelmien löytäminen ja niiden integrointi järjestelmiin ei välttämättä onnistu ilman ongelmia.

Testaukseen tarkoitettuja ohjelmistoja löytyy paljon ja ongelmaksi muodostuu yleensä valinnan tekeminen kaikista tarjolla olevista ohjelmista. Jo pelkkien ohjelmistojen koontityökalun valinta saattaa osoittautua mielenkiintoiseksi projektiksi, kun joudutaan kartoittamaan tarkasti, mitä tarvitaan ja mihin tarkoitukseen sitä tarvitaan. Loppujen lopuksi asioiden järjestelmällinen läpikäynti ja hyvä suunnittelu helpottavat asioiden toteutusta ja antavat hyvän kuvan projektin laajuudesta. Tärkeintä on ensin ymmärtää projektin tarpeet ja lopulta kokonaisuus.

Insinöörityön tavoitteena on parantaa ja nopeuttaa yrityksen testausmenetelmiä ottamalla käyttöön testausta auttavia työkaluja. Työn alussa esitellään testauksen aloittamiseen liittyvät tuotteen valintaprosessi sekä asiakasyrityksen testauksen alkutilanne.

Projektin alussa parannettiin jo käytössä olevia menetelmiä ja tutkittiin menetelmien muutosmahdollisuuksia. Projektin lopullisena tuotoksena saatiin Aspida Oy:lle uusi järjestelmä, joka sisältää testauksen eri osioiden automatisoinnin ohjelmistojen koonnista aina testien suorittamiseen asti.

Avainsanat: testaus, automatisointi, jatkuva integraatio, selenium, cruisecontrol

Metropolia

ABSTRACT

Name: Sami Sinisalo

Title: Implementation of Automatic Testing and Continuous Integration System

Date: 21.05.2010 **Number of pages:** 39

Department: Information Technology **Study Programme:** Software Engineering

Instructor: Auvo Häkkinen, Principal Lecturer

Supervisor: Chief Technology Officer (CTO), Mika Huhtamäki, Aspida Ltd.

Aspida Oy produces and executes interned-based software to analyze and portray data. Software testing is part of each of the company's software development projects. This way the company can assure and maintain the quality of their software and secure the functionality of their products. The automatisation of the software testing requires a lot of research and planning before the final execution. To find the right components, testing software and testing methods and to integrate them to the system is not necessarily completely trouble-free.

The great amount of existing testing software becomes a problem when choosing the right one to use. Simply choosing the right software composition tool becomes an interesting project as the needs and purposes of the tool are explored. In the end the systematic analyze and exhaustive planning facilitates the execution and gives a good image of the quality of the project. The most important thing is first to understand the needs and finally the whole project.

The thesis aims to improve and fasten the company's software testing methods. At the beginning the starting of the testing is introduced by presenting the process of choosing the testing software and presenting the current testing environment in the company. First the methods and tools already in use are enhanced and finally the company's testing process is presented containing the automatisation of different parts of the testing software from the software composition to finishing the tests.

Keywords: testing, automatization, continuos integration, selenium, cruisecontrol

SISÄLLYS

1 JOHDANTO 1

2 TESTAUSTILANTEEN KARTOITUSMENETELMÄ 2

3 YRITYKSEN TESTAUKSEN ALKUTILANNE 3

4 TYÖKALUOHJELMIEN VALINTAMENETELMÄ 7

4.1 Ohjelman valintaprosessi 7

4.2 Ohjelman vaatimusten tunnistus 9

5 UUDEN YMPÄRISTÖN JA TESTAUSPROSESSIN KUVAUS 12

6 OHJELMAT 14

6.1 CruiseControl 15

6.1.1 Jetty-käyttöoikeudet 19

6.1.2 Apache Ant 20

6.2 Shell-skriptit 23

6.3 Selenium 26

7 TESTIEN TEKEMINEN JA AJO 27

7.1 Selenium Java 32

7.2 Java-testit 32

7.3 Raportointi ja kuvankaappaus 36

7.4 Testit ja Xpath 36

7.5 Testidatan luominen Selenium-testillä 37

8 MAHDOLLISUUDET JA PARANNUSAJATUKSET 37

9 YHTEENVETO 38

VIITELUETTELO 39

LIITTEET

LIITE 1 SELENIUM IDE HTML -TESTITAPAUS LYHENNETTY ESIMERKKI

LIITE 2 SELENIUM IDE JAVA -TESTITAPAUS LYHENNETTY ESIMERKKI

LIITE 3 SELENIUM TESTI HTML SUITE -TIEDOSTO

LIITE 4 SELENIUM TESTI TULOS TIEDOSTO

LIITE 5 RAPORTOINTIA VARTEN LISÄTTÄVÄT ASETUKSET

LIITE 6 CRUISECONTROL CONFIG.XML

LIITE 7 PROJEKTIN BUILD.XML –TIEDOSTO

1 JOHDANTO

Nykymaailmassa tietokoneet ja internetin käyttö on lisääntynyt selvästi. Yritykset tuottavat entistä enemmän uusia ohjelmistoja ja internetpalveluita vastaamaan kasvavaa kysyntää. Yritykset pyrkivät siirtämään palvelujaan verkkoon ja tätä kautta saamaan yhteyden kuluttajiin. Kuitenkin ohjelmistojen ja palveluiden valmistaminen tapahtuu edelleen ihmisten toimesta ja erittäin tiukoilla aikatauluilla, jolloin ohjelmistoihin jää edelleen suuri määrä virheitä. Testauksella on ohjelmistojen laadunvarmistuksessa suuri rooli.

Ohjelmistotestauksen tärkeyttä ja osuutta ohjelmistojen laadussa ei voida liikaa korostaa. Ohjelmistotestaus on yksi kriittisimmistä osista puhuttaessa ohjelmistojen laadusta. Ohjelmistojen virheiden aiheuttamat kustannukset ovat yksi suurimmista testauksen puolestapuhujista. [1, s. 464.]

Tämän insinöörityön aiheena on tutkia, millä tavoin Aspidan nykyisiä ohjelmistotestausmenetelmiä ja -tapoja voitaisiin parantaa automatisoimalla eri testausvaiheita mahdollisimman pitkälle. Työssä pohditaan, mitä voidaan automatisoida ja miten, mitä tulee ottaa huomioon, kun parannuksia tehdään, ja mitä pitää parantaa. Työ kattaa kohdat, jotka kuuluvat automatisoinnin piiriin suunnittelusta, ongelmien ratkomisesta ja toteutuksesta lähtien. Työssä pyritään vastaamaan kysymyksiin: mikä on nykyisen laitteiston rakenne ja käytettävät menetelmät, mitkä ovat tuotekehityksen kohteet ja miten kehittäminen toteutetaan?

Opinnäytetyössä selvitettiin ja toteutettiin Aspida Oy:lle automaattinen testausjärjestelmä, jolla voidaan jatkossa testata yrityksen kehittämiä ohjelmistoja. Tarkoitus oli toteuttaa kattava ja helppo rakenne ja parantaa nykyisiä yrityksen testaus- ja ohjelmanhallinnointimenetelmiä. Työssä kartoitettiin käytössä olevien ohjelmien käyttömahdollisuuksia ja toteutustapoja, sekä pyritään toteuttamaan niillä mahdollisimman joustava ja kattava kokonaisuus. Työ sisältää perustoteutuksen lisäksi yleistä pohdintaa, ongelman ratkaisua ja käytännön kuvauksen automaattisen järjestelmän pystytyksestä aina käyttöönottoon asti. Tarkoituksena oli saada laaja yleiskuva ohjelmien muuntautumiskyvystä, luotettavuudesta, käytettävyydestä ja tarpeellisuudesta yleisissä ohjelmistoprojekteissa.

Testiprojekteina käytettiin Aspida Oy:n kehittämiä web-palvelinsovelluksia, jotka integroitiin järjestelmään työn edetessä.

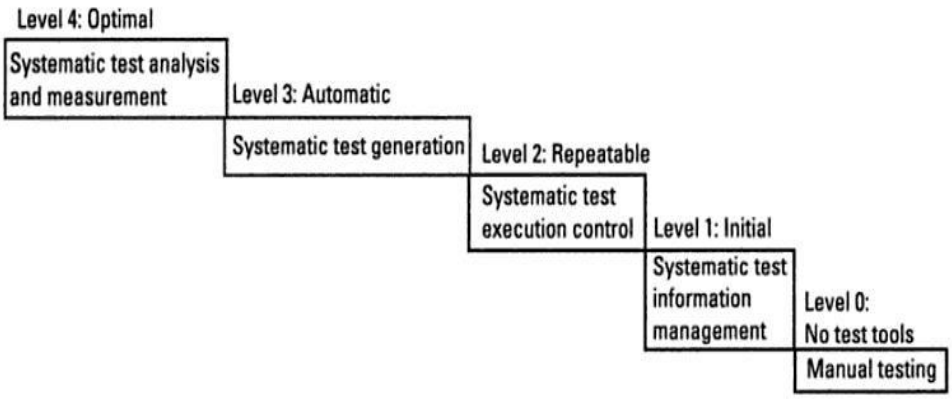
Työn alussa (luku 2) esitellään testaamistilanteen kartoittamista yrityksessä. Tämän jälkeen kuvataan yrityksen testauksen tilannetta alussa (luku 3). Seuraavaksi kerrotaan valintaprosessimenettelystä ja vaatimusten tunnistamisesta (luku 4). Uudesta ympäristöstä ja testausprosessista kerrotaan luvussa 5, jonka jälkeen esitellään käyttöön valittuja ohjelmia ja niiden ominaisuuksia (luku 6). Loppupuolella käydään läpi testien tekemistä ja niiden suorittamista (luku 7). Viimeiseksi kerrotaan vielä järjestelmän mahdollisuuksista ja parannusajatuksista (luku 8).

2 TESTAUSTILANTEEN KARTOITUSMENETELMÄ

Testauksen tarkoituksena on löytää ohjelman mahdolliset virheet ja toimintaongelmat ennen ohjelman varsinaista käyttöönottoa. Hyvällä testillä on suuri todennäköisyys löytää ohjelmasta virhe. Testien tulisi olla myös nopeita ja vaivattomia suorittaa, jolloin testejä voidaan tehdä useammin ja tulosten vertailu olisi helppoa. Testaamisella voidaan myös todeta ohjelman toimivan halutulla tavalla ja nähdä, toteutuvatko kaikki ohjelmalle asetetut tavoitteet ja vaatimukset. Pääsääntöisesti testaamisella ei pystytä osoittamaan virheiden puuttumista, vaan testaamisella pystytään osoittamaan virheiden olemassaoloa. [1, s. 465 – 466.]

Automaattisella testauksella tarkoitetaan yleensä normaalin käsintehdyn testauksen korvaamista automatisoidulla testauksella. Tällöin siihen sisällytetään kaikki yleiset testausmenetelmät, testausstrategiat ja ratkaisut, joita on aiemmin käytetty normaalissa käsintestauksessa. Päämääränä on vapauttaa työntekijöitä rutiinitestauksesta, nopeuttaa testaamista sekä nostaa testaamisen tasoa ja laatua. Automatisointi ei välttämättä vaikuta suoraan testauskustannuksiin, vaan automatisointi näkyy yleisesti välillisenä hyötynä eri projekteissa – projektit pysyvät yleensä paremmin aikataulussa ja projektin tuottamat tuotteet tulevat laadukkaammiksi [2, s. 158 - 159]. Pelkkä ohjelmiston testauksen automatisointi ei automaattisesti paranna tuotteen laatua ja nopeuta testausta, vaan automatisoinnin yhteydessä on otettava tarkasti huomioon miten, automatisointi tulisi toteuttaa, jotta siitä saataisiin kaikki mahdollinen hyöty irti.

Yrityksen testauksen tilanne tulee kartoittaa ennen automaattisen työkalun valintaa. Tähän kartoittamiseen voidaan käyttää esimerkiksi alla olevaa Gaon, Tsaon ja Wun esittämää mallia yrityksen testaustilanteesta. Kuvassa 1 on kuvattu viidellä tasolla testauksen eri tilat yrityksessä.

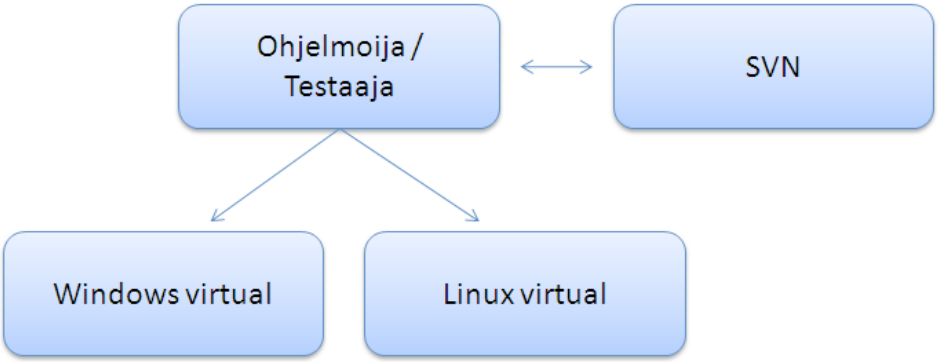


Kuva 1. Testauksen tasot [2, s. 160]

Ensimmäisellä tasolla (level 0) ei testaustyökaluja käytetä, vaan kaikki testaus tehdään käsin alusta alkaen. Toisella tasolla (level 1) testaus on selvästi järjestelmällisempää, ja suuri osa testauksen vaatimuksista on otettu huomioon, mutta mitään apuvälineitä ei ole vielä otettu käyttöön testauksen toteutuksessa tai suunnittelussa. Kolmannella tasolla (level 2) testaus on erittäin järjestelmällistä sallien tekijöiden toteuttaa testejä ja tarkastella tuloksia helposti ja tarkasti. Neljännellä tasolla (level 3) testaus on automaattista, mutta mitään tarkempia tuloksia testien kattavuudesta ja tehokkuudesta ei vielä saada. Viimeisellä tasolla 5 (level 4) testaus on optimoitu loppuun asti ja automaattinen testausympäristö tuottaa mittaustuloksia [2, s. 159 - 160].

3 YRITYKSEN TESTAUKSEN ALKUTILANNE

Ohjelmistojen testausta suoritettiin Aspida Oy:ssä alun perin ohjelmoijien toimesta, jolloin jokainen ohjelmoija testasi oman, sekä muiden ohjelmanosat tarpeen mukaan (kuva 2).



Kuva 2. Aikaisempi testaustilanne

Kuvassa 2 on kuvattu tilanne testauksen tapahtumisesta, jossa ohjelmoija eli testaaja on yhteydessä svn-tietovarastoon, josta saadaan aina ohjelman uusin versio. Ohjelma voidaan testata testaajan omalla koneella tai testaaja voi käyttää virtuaalista Windwos–palvelinta tai Linux–palvelinta testauksen tekemiseen. Virtuaalipalvelimia käyttäessä testaaja joutuu kasaamaan ohjelman ja asentamaan sen palvelimelle itse omalla koneeltaan. Tällöin jokaisen kasauksen lopputulos voi olla erilainen riippuen kasaajan koneesta ja kasaustyökalusta, jolloin testatun ohjelman toiminta ei välttämättä vastannut ohjelman toimintaa tuotannossa. Ohjelman valmistuksen loppuvaiheessa koko ohjelmalle tehtiin aina suurempi ja kattavampi testaus kaikkien ohjelmoijien toimesta.

Edellä mainittuihin ongelmiin haettiin parannusta käyttämällä testaukseen erillistä henkilöä, joka vastaisi vain testauksen toimimisesta ja toteutuksesta. Kaiken tekeminen käsin vaati kuitenkin aikaa ja tarkkuutta, jolloin usean ohjelmistoprojektin käsittelyssä sattui inhimillisiä virheitä. Virheiden, kunnollisen raportoinnin ja ajankäytön tehostamisen takia päätettiin uudistaa toimintatapoja ja -menetelmiä. Koska kunnollisia valmiita toimintatapoja ei ollut, kaikki piti toteuttaa ja suunnitella alusta alkaen.

Aluksi testaus toteutettiin käsin tekemällä testitapaukset Excel-tiedostoon, jossa kaikki testit oli listattu yhdellä sivulla (kuva 3).

Test id	Group	Case name	Description	Expected result	Pass	Comments
LOG001	Login	Login	1. Log in using the username and password given by the test manager	Project listing appears		
LOG002	Login	Logout	1. Click Log out button	Login page appears		
LOG003	Login	Login with invalid username / password	1. Try to log in using a random username 2. Try to log in using valid username and invalid password	Red text "The username and password are not valid" appears		
BP001	Briefing Package	Create new project	1. Login 2. Click Create New Project button 3. Fill in the form: Name: testiprojekti Description: just a test Deadline: 2007/10/25 Name: Testifirma VAT-Number: Add a project team member: <your username> Click Save			
BP002	Briefing Package	View Briefing package	1. Select briefing package in the navigation menu	Briefing package displays correct data There is a Modify button on bottom right		
BP003	Briefing Package	Edit Briefing package	1. Select briefing package in the navigation menu 2. Click Modify 3. Add some text in all the fields 4. Click Save	All the changes are shown in the Briefing Package		

Kuva 3. Alkuperäinen testitiedosto

Testejä parannettiin jakamalla kaikki testit omiin asetus-välilehtiin (setup sheet) (kuva 4).

Comprehensive test

Project

Nmb	Function	Test	Result	Accepted	Comment
	Project	1. In 'Username' field write "test@mail.com", in 'Password' field write "test" and click 'Login' 2. Click 'Edit project template' button 3. Click 'Add new template' button 4. Write in new template 'Name' field "TestTemplate" 5. Click 'TestTemplate' 'Master index' section 'Import' button, click 'Browse...', select 'MasterIndex2.xls' file and open it 6. Click 'Import' button and after that click 'Close' button 7. Click TestTamplate 'Report' section 'Import' button, click 'Browse...' choose 'Report2.rtf' and open it 8. Click 'Import and then 'Close' button 9. Click 'Save'	2. You should see only one template and no Automatic checkboc active 6. After using Import button New blue text 'File imported!' appears 8. After using Import button New blue text 'File imported!' appears New Tamplate TestTemplate was added		
		1. Cilck 'Edit project template' button 2. Click 'Add new template' button 3. Write in new template 'Name' field "Template" 4. Click 'TestTemplate' 'Master index' section 'Import' button, click 'Import' button 5. Click 'Browse...', select 'MasterIndex2.xls' file and open it 6. Click 'Import' button and after that click 'Close' button 7. Click TestTamplate 'Report' section 'Import' button, click 'Import' 8. Click 'Browse...' choose 'Report2.rtf' and open it 9. Click 'Import and then 'Close' button 10. Click 'Remove field' button in 'Tamplate' line	4. You will get red message 'Please select a valid file!' 7. You will get red message 'Please select a valid file!' 10. 'Template' template disappears		

Kuva 4. Korjattu testitiedosto -esimerkki

Välilehdellä kerrottiin testissä tarvittavat alkuvalmistelut ja kerrottiin tekijälle, mitä lisätiedostoja tarvitaan. Seuraavassa välilehdessä oli yhteenvetotaulukko kaikkien testien tuloksista, joista kävi selkeästi ilmi testien onnistuminen. Tämän jälkeen jokainen välilehti sisälsi testejä koskevia tietoja, jotka oli jaoteltu testityypin eli testitapausten mukaan esimerkiksi: käyttäjän hallinta, projektin hallinta ja raportin hallinta –osioihin.

Kaikissa testeissä on hyväksyntä-kohtaan (Accepted) laitettu lajittelu, jolla pystytään nopeasti erittelemään saadut tulokset. Kaikki saadut tulokset voidaan myös nähdä tulostaulusta (kuva 5), jossa jokaisen testattavan toiminnon tulokset on eritelty omaan taulukon kohtaan onnistuneisiin (Accepted), virheellisiin (Incomplete) tai keskeneräisiin (Not started).

Comprehensive test
Project

	Customer	Project	BPB	MI	Pr
Accepted	2	0	0	0	0
Incomplete	1	0	0	0	0
Not Started	2	4	8	7	7
Tests	5	4	8	7	7

Kuva 5. Testien tulostaulu

Testien kehittelyn yhteydessä huomattiin, miten hankalaa testejä on tulkita nopeasti ja huomata virheelliset kohdat. Tähän saatiin ratkaisu Excelin tarjoamasta värikoodaus- ja filtteröinti-toiminnosta, jolla pystyttiin helpottamaan tulosten lukemista ja tulkintaa (kuva 6).

Comprehensive test
Project

Nmb	Function	Test	Result	Accepted	Comment
Cu001	Login	1. Open your wer browser address in Setup sheet 2. In 'Username' field write "test", in 'Password' field write "test" and click 'Login' 3. In 'Username' field write "test2@mail.com", in 'Password' field write "test" and click 'Login'	2. You should see red message 'The username and password are not valid' 3. You will see a new window "CustomerView"	Not Started	
Cu002	Company	1. Click 'Add customer' button 2. In 'Name' field write "Test" and click 'Save' button	2. New customer was added under Name field	Accepted	
Cu003	User	1. Click 'Edit' button in customer 'Test' line 2. Add "Test" in 'ADD USER' 'Name' field, add "test@mail.com" in 'Username' field, add "test" in 'Password' field, add "Test" in 'Initials' field and click 'Save' 3. Add "Test1" in 'ADD USER' 'Name' field, add "test1@mail.com" in 'Username' field, add "test1" in 'Password' field, add "Test1" in 'Initials' field and click 'Save' 4. Change 'Admin' to 'User' in 'Group' drop-down menu, add "Test3" in 'ADD USER' 'Name' field, add "test3@mail.com" in 'Username' field, add "test3" in 'Password' field, add "Test3" in 'Initials' field and click 'Save'	2. You will see "Test" user under 'USERS' heading. User should be an Admin 3. User "Test1" has added as an Admin 4. User "Test3" has added as an User	Incomplete	

Kuva 6. Korjatun testitiedoston esimerkki

Uuden menettelyn tuoman muutoksen johdosta testeistä tuli toistettavampia ja vähemmän virhealttiita inhimillisille virheille kuin aikaisemmin. Tämä ei kuitenkaan riittänyt, vaan rutiinitesteistä piti saada nopeampia ja varmempia toteuttaa kuin ihmisen käsin tehdyt testaukset. Vaihtoehtona tuli esiin automatisoitu testaus, jossa testit luodaan alussa käsin ja sitten ohjelman avulla. Testit tuli pystyä suorittamaan useaan kertaan. Niiden piti olla toistettavia ja niistä tuli saada lopuksi halutunlainen raportti testien tuloksista. Kaikki vaikutti kaiken kaikkiaan hyvältä, kunnes ymmärrettiin automaattisen testauksen vaatimat muut valmisteluvaiheet, jotka piti saada toteutettua automaattisesti. Muita vaiheita testauksen lisäksi on niin kutsuttu jatkuva integraatio, jossa kehitettäviä ohjelmia hallitaan eri ohjelmien avulla. Ohjelman koonti (build), asentaminen (installation), asetustiedostojen (configs) käsittely ja varmuuskopioiden (backups) ottaminen kannattaa suorittaa ennen ohjelman testaamista, jolloin päästään eroon rutiininomaisesta käsityöstä. Tällöin päätettiin koota pelkkää testausta kattavampi kokonaisuus ja ottaa samalla huomioon kaikki testausta edeltävät ja sen jälkeen tulevat osa-alueet.

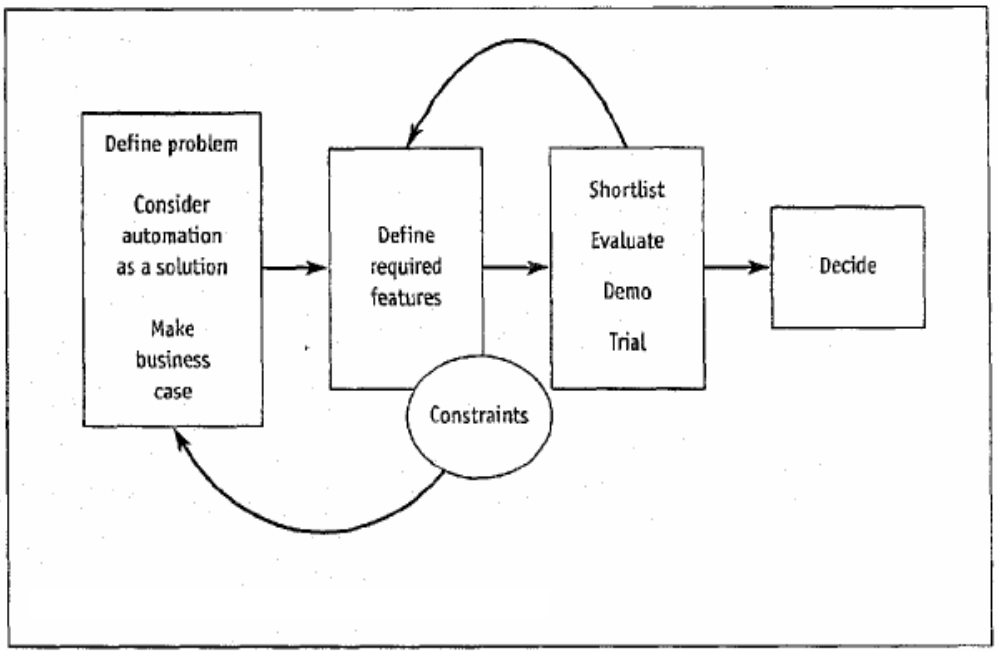
4 TYÖKALUOHJELMIEN VALINTAMENETELMÄ

Työkaluohjelmien valinnassa tulee olla erittäin tarkka, jotta valituksi tulee parhaiten tarkoituksiin sopivat ohjelmat. Ohjelmien tarkoitus voi olla esimerkiksi ohjelmiston kasaus tai ohjelmiston käyttöliittymän testaus, jotka asettavat ohjelmalle omat vaatimuksensa. Vaatimukset kannattaa määritellä huolella, sillä jos jokin tärkeä kriteeri on unohdettu, se huomataan yleensä vasta projektien loppupuolella, kun kokonaisuus on jo kasassa. Ohjelmien valinnassa kannattaa muistaa, että se vaikuttaa lopussa koko kokonaisuuteen ja toimii perusrakenteena koko projektille. Valintakriteereiden tunnistaminen ei välttämättä ole helppoa ja joissakin tapauksissa se saattaa aiheuttaa ongelmia, mutta useimmiten tässäkin voidaan käyttää vakiintuneita tapoja vaatimusten määrittämiselle.

4.1 Ohjelman valintaprosessi

Valintaprosessilla on tarkoitus valita yrityksen toimintaan ja tarpeisiin sopiva ohjelma tai ohjelmat ja helpottaa valinnan tekemistä suuresta määrästä ohjelmia. Tällöin voidaan varmistaa, että ohjelmasta saadaan irti mahdollisimman suuri hyöty [3, s. 248]. Valintaprosessissa tulee ottaa

huomioon myös ohjelman tulevien käyttäjien määrä ja taitotaso. Jos ohjelman käyttäjiä, on useita, tulee valintaprosessista laajempi ja tarkempi, kun taas vain muutamalla käyttäjällä jolloin valinta voidaan tehdä nopeammin ja valinnassa voidaan käyttää suppeampaa tietämystä. Laajemmassa ja suppeammassa valintaprosessissa voidaan käyttää samaa valintaprosessia, vaikka valinnan tarkkuus tai vaatimukset eivät olisikaan samanlaisia. Kuvassa 7 on selkeä kuvaus valintaprosessin etenemisestä aina ongelman löytämisestä päätöksentekoon asti. [3, s. 249.]



Kuva 7. Valintaprosessin kuvaus [3. s. 251]

Alussa ensimmäisessä vaiheessa määritellään ongelma (Define problem), johon on tarkoitus löytää ratkaisu. Seuraavaksi mietitään automaattista ratkaisua ongelman ratkaisemiseksi (Consider automation as a solution) ja tehdään tarvittaessa kustannuskartoitus (Make business case). Seuraavaksi edetään toiseen määrittelyvaiheeseen, jossa tarvittavat vaatimukset ja ominaisuudet kartoitetaan (Define required features). Tarvittaessa voidaan palata takaisin tarkentamaan ensimmäisen vaiheen päätöksiä. Palaaminen takaisin ensimmäiseen vaiheeseen johtuu yleensä määrittelyiden asettamista rajoituksista, jolloin joudutaan miettimään automaation tärkeyttä tai kustannuksia. Toisesta vaiheesta edetään kolmanteen vaiheeseen eli arviointivaiheeseen, jossa voidaan kokeilla tuotetta. Tästä voidaan tarvittaessa palata takaisin tarkentamaan vaatimuksia ja ominaisuuksia.

Kolmannen vaiheen jälkeen siirrytään neljänteen eli päätösvaiheeseen, jossa tehdään päätös valitusta ohjelmasta.

Ohjelman valinnassa ei kannata lähteä liikkeelle suoraan tutkimalla kaupallisia ohjelmia, vaan ensin tulisi selvittää tarpeiden asettamat vaatimukset. Tarpeita voivat olla ohjelman helppokäyttöisyys, toimintavarmuus tai käyttöjärjestelmäriippuvuus. Mikäli valitaan vääränlainen tai toiminnallisesti puutteellinen ohjelma, voi edessä olla ongelmia ja hankaluuksia saada työkalu asennetuksi tai toimimaan halutulla tavalla. Ohjelman käyttäjät saattavat pitää ohjelman käyttöä hankalana ja lopettaa ohjelman käytön sen puutteellisuuden takia. Alussa pienet toiminnallisuuksien puutteet vaikuttavat pieniltä ongelmilta, mutta käyttöönoton jälkeen yleensä huomataan, ettei ohjelma toimikaan halutulla tavalla. [3, s. 249.]

Valintavaiheen alussa kannattaa ottaa mukaan ohjelman tulevat käyttäjät ja keskustella heidän kanssaan valintaan liittyvistä kriteereistä. Näin vältetään tilanne, jossa käyttäjät ilmoittavat, etteivät he ole saaneet haluamaansa ohjelmaa ja kieltäytyvät käyttämästä sitä. Tällainen tilanne tapahtuu yleensä silloin, kun yksittäisen ihmisen tekemä päätös vaikuttaa useaan ihmiseen. Isommissa yrityksissä voidaan ryhmän johtoon laittaa yksi ihminen, jonka tulee pystyä kokoamaan ryhmä osaavia ihmisiä, jotka pyrkivät ottamaan kaikkien näkemykset huomioon saavuttaakseen parhaan mahdollisen tuloksen. [3, s. 250 - 251.]

4.2 Ohjelman vaatimusten tunnistus

Yleensä vaatimuksia uudelle esimerkiksi automaatitestausohjelmalle on useita. Osa liittyy testauspuolen ongelmiin joita halutaan ratkoa ja osa taas liittyy teknisiin tai ei-teknillisiin vaatimuksiin. Nämä työkalulle asetetut vaatimukset pitää tunnistaa heti aluksi, jotta voidaan paremmin verrata eri ohjelmia keskenään. [3, s. 252.]

Alussa tunnistetaan ratkaistava ongelma: mihin ongelmaan ohjelma voisi tuoda ratkaisun? Tunnistamalla tarkasti ongelmakohdat, jotka halutaan ratkaista, ei yksin riitä, jos kartoitusta tekee suurempi ryhmä. Tällöin koko ryhmän pitää saada kaikki sen jäsenet ymmärtämään ne samalla tavalla, jolloin niitä voidaan käyttää myöhemmin valintavaiheen kriteereinä. Ongelmina voi olla esimerkiksi käsin tehdyssä testauksessa tapahtunut

virhe, muutosten aiheuttamat aikaongelmat, testitapauksissa aiheutuvat virheet, riittämätön dokumentaatio testeistä, tietämättömyys ohjelman testaamisen laajuudesta ja tehoton testaus. Kaikkia näitä ongelmia ei välttämättä voida pelkästään ratkaista automatisoinnilla, joten ongelmista on syytä laatia lista ja järjestää ne tärkeysjärjestykseen. [3, s. 253.]

Käsin tehdyt testit ovat yleensä työläitä, aikaa vieviä, epäjohdonmukaisia, tylsiä ja pitkäkestoisia. Myös tulosten vertaaminen on erittäin virhealtista. Ensisilmäykseltä tulosten vertaaminen vaikuttaa loistavalta kohteelta muuttaa automaattiseksi, ja se voikin olla totta suuremmassa osasta tapauksista. Automatisointi ei kuitenkaan ole ainoa ratkaisu manuaalisen testauksen aiheuttamiin ongelmiin. Tärkeimpänä voidaan pitää sitä, miten yritys näkee käsin tehdyn testauksen arvon verrattuna automatisoituun testaukseen ja mikä on näiden kahden kustannusero. Mikäli testit ovat tulleet liian pitkäkestoisiksi suorittaa, niitä voitaisiin karsia tai poistaa kokonaan, jolloin jäljelle jäävät vain tarpeelliset testit. Tällöin testien tekemiseen käytettävää kokonaisaikaa saadaan lyhennettyä ja testit voidaan toteuttaa nopeammin ja tehokkaammin. Tilanteessa, jossa aikaa kuluu liikaa testien toteuttamiseen eikä niitä voida mitenkään enää supistaa, voidaan aina miettiä lisähenkilöstön palkkaamista testien toteutukseen. Jos taas yksittäisten testien suorittaminen on raskasta ja hankalaa, voidaan yksittäisiä testejä muokata joko lyhyemmiksi tai selvästi helpottaa niiden toteuttamista. Tällöin manuaalisesta testauksesta tulee tehokkaampaa ja tuottoisempaa. [3, s. 253.]

Yleensä saatujen tulosten vertaaminen odotettuihin tuloksiin manuaalisesti aiheuttaa virheitä. Ongelmana voi olla testaajien ymmärtämättömyys testimenettelyistä tai se, miten testaajat on koulutettu hallitsemaan käytettäviä testausmenetelmiä tai laitteita. Onko tietojen syöttäminen, testien toteuttaminen ja tulosten analysointi toteutettu oikein ja ovatko testaajat tietoisia siitä mitkä ovat oikeita tuloksia. Testitulosten vertaaminen koneellisesti on hyvä kohde automatisoinnille ja samalla virhealttius pysyy erittäin alhaisena. Siksi monet työkalut sisältävätkin vertailuun tarkoitetun ohjelman osan. [3, s. 254.]

Tilannetta, jossa testaaminen muuttuu tylsäksi, voidaan pitää viitteenä siihen, että aputyökalu on tarpeen. Asiat, jotka ihmiset tuntevat tylsiksi,

voidaan useimmiten toteuttaa paremmin, tarkemmin ja tehokkaammin koneella. [3, s. 254.]

Automaattinen testaus on hyvä tapa toteuttaa regressiotestaamista. Kun automaattinen järjestelmä on saatu valmiiksi, niin regressiotestaus voidaan suorittaa nopeammin ja tehokkaammin kuin manuaalisesti. Tärkeämpää on kysyä, miksi regressiotestaamista ei ole alun perin hoidettu kunnolla. Onko kyse siitä, ettei aika riitä kunnolliseen regressiotestaamiseen, vai siitä, että ohjelmaan on tehty viimehetken muutoksia, joita ei ole vielä otettu huomioon automaattisen testauksen puolelle. Tällöin on helpompaa ja nopeampaa tehdä testit manuaalisesti kuin alkaa korjata automaatista järjestelmää. Järjestelmässä on kuitenkin voitu ottaa huomioon se, että pienet ohjelman muutokset eivät haittaa testien ajoa ja testit voidaan suorittaa automaattisella järjestelmällä ilman mitään ongelmia. Uudet muutokset testataan manuaalisesti, jolloin testit voidaan lisätä myöhemmin automaattisen järjestelmän puolelle, kun aikataulu sen sallii, eikä automaattisesta järjestelmästä tule ylimääräistä murhetta sen käyttäjille. [3, s. 254.]

Automatisoitu testaustyökalu raportoi siitä, mitä kaikkia testejä on ajettu ja mitä tuloksia niillä on saatu. Testauksen laajuus jää usein kertomatta eikä tarkkaa tietoa siitä miten laajasti ohjelmaa on testattu ole tiedossa. Tähän on olemassa omat mittausohjelmansa, mutta ne eivät välttämät sisälly testausohjelmaan mukaan. Laajuus pystytään helposti mittaamaan kuitenkin manuaalisesti listaamalla esimerkiksi ohjelman kaikki toiminnot ja tarkistamalla, että testi käy kaikki kohdat läpi. Puuttuville toiminnolle voidaan tarvittaessa tehdä samalla omat testit ja lisätä ne muiden jatkoksi. [3, s. 255.]

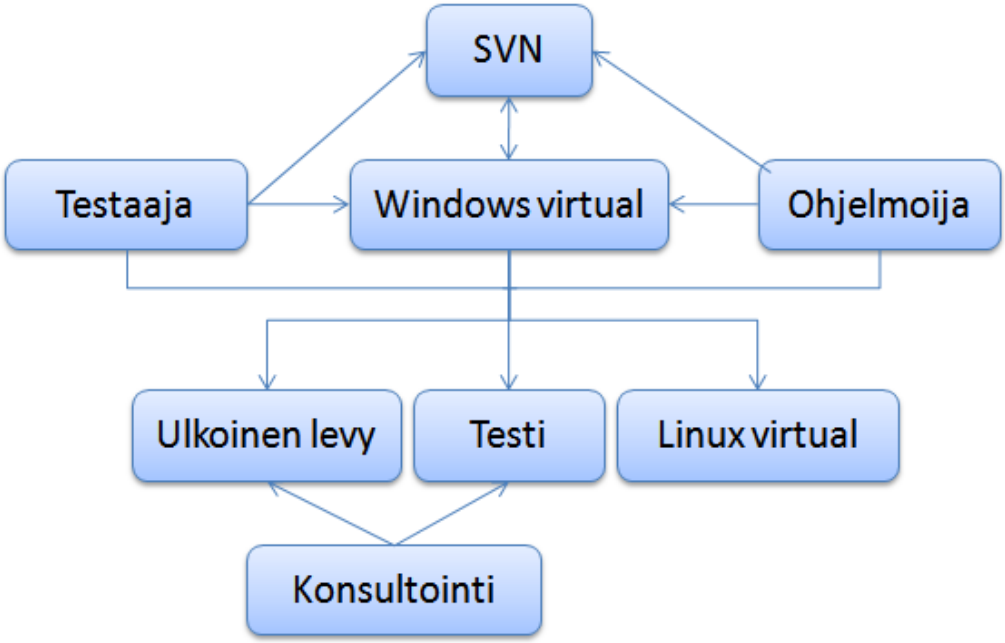
Testattavan ohjelman virheet tulee löytää ennen varsinaisia käyttäjiä. Jos testaus ei löydä näitä virheitä, niin yleensä kyse on huonosti suunnitelluista testitapauksista eikä siitä, onko kysymyksessä automaattinen tai manuaalinen testaus. Jos taas kysymys on siitä, että aika ei riitä manuaaliseen testaukseen, voi automaattisesta työkalusta olla apua. Testitapausten laatu kuitenkin viimekädessä vaikuttaa siihen, miten hyvin ohjelman virheet löydetään, eikä automaattisella testauksella ole vaikutusta testien laatuun suunnittelun osalta. [3, s. 255 - 256.]

Kun testaustyökalun tarpeellisuudesta ja hankkimisesta on päätetty, edetään pohtimaan, mikä on paras mahdollinen aika hankkia tarvittavat aputyökalut. Vaikka tarve olisi, aina aika ei välttämättä ole sopiva: yrityksessä on kiire, vastuuhenkilöä ei ole valittu, ihmiset ovat tyytyväisiä nykyiseen testausmenettelyyn tai siihen ei ole annettu lupaa ylemmältä johdolta. Täytyy muistaa, ettei automatisoitu ympäristö korjaa huonoja testausprosesseja. Prosesseja voidaan ja päästäänkin muuttamaan asennuksen yhteydessä. Tällöin on myös hyvä hetki parantaa muita testaukseen liittyviä toimintatapoja. Tämän kaiken keskellä tulee muistaa, että tämä kaikki tulee vaatimaan ja viemään paljon aikaa ja rahaa. [3, s. 257.]

Miten paljon automatisoidusta järjestelmästä tulisi olla apua, jotta se kannattaa ottaa käyttöön? Tätä voidaan mitata erilaisilla muuttujilla, kuten esimerkiksi testiajojen pituudella. Testiajoissa mitataan, miten paljon manuaalisesti suoritettavat testit vievät aikaa ja verrataan tuloksia niistä aiheutuviin kustannuksiin. Testien suoritusaikaa manuaalisesti voidaan helposti verrata odotettuun ajoaikaan automatisoidun ajoajan kanssa ja nähdä, miten suuri hyöty tästä voidaan mahdollisesti saada. [3, s. 257.]

5 UUDEN YMPÄRISTÖN JA TESTAUSPROSESSIN KUVAUS

Uusi ympäristö koostuu vanhoista virtuaalikoneista ja muutamasta uudesta järjestelmän osasta, kuten ulkoisesta kovalevystä ja erillisestä ulkoverkossa sijaitsevasta koneesta. Ympäristöä on laajennettu tarpeiden mukaan ottamalla huomioon työntekijöiden ja testauksen asetamat vaatimukset. Työntekijöiden vaatimuksina oli esimerkiksi helppo yhteys koottuihin paketteihin, niin sisäverkosta kuin ulkoisesta verkostakin. Ulkoisen verkon vaatimus tulee konsultoinnista, joka tapahtuu työpaikan ulkopuolisissa toimitiloissa asiakkaan luona. Kuvassa 8 on kuvattu uusi ympäristö.



Kuva 8. Uuden ympäristön kuvaus

Uusi ympäristö koostuu testaajan omasta koneesta (Testaaja), ohjelmoijien omista koneista (Ohjelmoija) ja yhteisestä svn –tietovarastosta (SVN). Järjestelmä sisältää useita Windows-virtuaalikoneita (Windows virtual) ja 3 Linux-virtuaalikonetta (Linux windows), jolla pääasiallinen testaus toteutetaan. Jatkuva integraatio-ohjelma on sijoitettu yhteen Windows virtuaalikoneeseen, josta ohjelma pääsee käsiksi kaikkiin tarvittaviin järjestelmän osiin. Ulkoinen kovalevy (Ulkoinen levy) on lisätty säilytystilaksi koontipaketeille. Ulkoisessa verkossa sijaitseva kone (Testi) on lisätty ulkoisen kovalevyn tapaan paikaksi, jossa pystytään testaamaan koottuja ohjelmia ja johon päästään käsiksi ulkoisesta verkosta.

Testaaja hallinnoi kaikkia koneita joko Virtual Desktop-yhteydellä tai SSH-yhteydellä. Tiedoston siirroissa voidaan käyttää apuna ftp-apuohjelmia kuten Filezillaa. Testaamisessa käytetään Linux- ja Windows-koneita riippuen tuotteen tarpeista ja vaatimuksista. Tuotteen lopullinen asennuspaikka tulee ottaa huomioon kun testausta suunnitellaan, jotta ympäristö olisi mahdollisimman samankaltainen kuin ohjelmiston tuotantoympäristö.

Ohjelmiston testaamista varten rakennettiin jatkuvan integraation ympäristö, johon kuuluu kolme Linux- ja useita Windows-koneita, joista kaikkia Linux-koneita ja Windows-koneita käytetään kasattujen ohjelmien asentamiseen ja niiden testaamiseen. Laitteistosta löytyy niin 64- kuin 38-bittisiä koneita.

Linux-käyttöjärjestelmänä käytetään CentOS RedHat-pohjaista jakelupakettia ja Windows-puolella XP, Vista tai Windows server-järjestelmiä. Tietokantoina löytyy perinteiset MySQL- ja MSSQL 2005 -tietokannat. Muita tarvittavia ohjelmia on Tomcat 6 -ohjelma ja sille Java-tuki.

Koneiden väliseen kommunikointiin käyttäjät käyttävät yleistä SSH -yhteyttä tiedostojen kopiointiin ja komentojen antamiseen palvelimelle. Tämä voidaan tehdä joko automaattisella tunnistuksella, kuten avaimella, tai kirjoittamalla käyttäjänimi ja salasana. Tietokantojen ja asennusten tutkimiseen käytetään SSH -yhteyttä ja sillä luotua tunnelia koneiden välille. Tämä siksi, että tietokantaportti oli yleisesti käytössä vain palvelimen sisällä.

Prosessiin kuuluu versionhallinta, jossa säilytetään kaikkien projektien koodit ja muut osat. Versionhallintajärjestelmään tehdyt muutokset laukaisevat CruiseControllerin projektien päivityksen ja koonnin (liite 6). Aluksi CruiseControl päivittää projektin tiedostot uusimpiin versioihin, jonka jälkeen se suorittaa projektien build.xml (liite 7) -tiedostoja, jolla kootaan projekti. Projektin koottu versio kopioidaan julkaistavaksi ja testipalvelimelle testausta varten. Build.xml lopussa käynnistetään käyttöliittymätestit, joiden tulokset tallennetaan julkaistavan version kanssa samaan paikkaan.

6 OHJELMAT

Projektin koontiohjelmaksi valittiin CruiseControl-ohjelmisto, joka tarjoaa luotettavan tavan koota ohjelmistoja sekä monipuolisia menetelmiä erityyppisten projektien koontiin. CruiseControl-ohjelmisto toimii vakiona Jetty HTTP -palvelimessa, jota voidaan tarpeen mukaan muokata helposti eri tarpeisiin. Ohjelmiston koonti ja kootun paketin muu hallinta tapahtuu Ant-työkalulla. Ant on monipuolinen työkalu, jossa löytyy erittäin laaja valikoima eri tarpeisiin sopivia toimintoja kuten scp-siirto (Secure Copy Protocol) paketeille. Koska testauksessa on käytössä Linux-palvelimia, käytössä on shell-skriptejä, joilla voidaan helposti hallita testien alkuvalmisteluita. Itse ohjelman testausohjelmaksi on valittu Selenium-tuoteperhe, joka tarjoaa laajan kokonaisuuden erilaisia testausohjelmia ja tuen eri ohjelmointikielille.

6.1 CruiseControl

CruiseControl on Open Source OSI sertifioitu jatkuvan integraation ohjelma, jolla kasataan ja hallitaan ohjelmistoja. Ohjelma toimii Jetty-webserverin päällä. CruiseControlin mukana tulee Apache Ant, jolla pystytään kokoamaan ohjelmistoprojekteja. CruiseControlin suurin hyöty on sen helppo käyttöönotto Windows-palveluna, jolloin ohjelma saadaan toimimaan suoraan yhdellä asennustiedostolla. CruiseControl:n helppo konfigurointi auttaa käyttöönotossa ja helpottaa aloittelijan edistymistä. CruiseControl-projekteja ohjataan config.xml-tiedostolla, jolla kutsutaan Apache Antin käyttämiä projektinkoonti build.xml-tiedostoja. Kasattu projektipaketti voidaan julkaista suoraan webkäyttöliittymässä. Koottu paketti voidaan myös kopioida haluttuun hakemistoon tai osoitteeseen. Ohjelmassa on pakotettu koonti (Force build)-toiminto, jolla voidaan pakottaa projektin koonti sitä tarvittaessa tai kasaus voidaan suorittaa yöllä ajastulla toiminnolla.

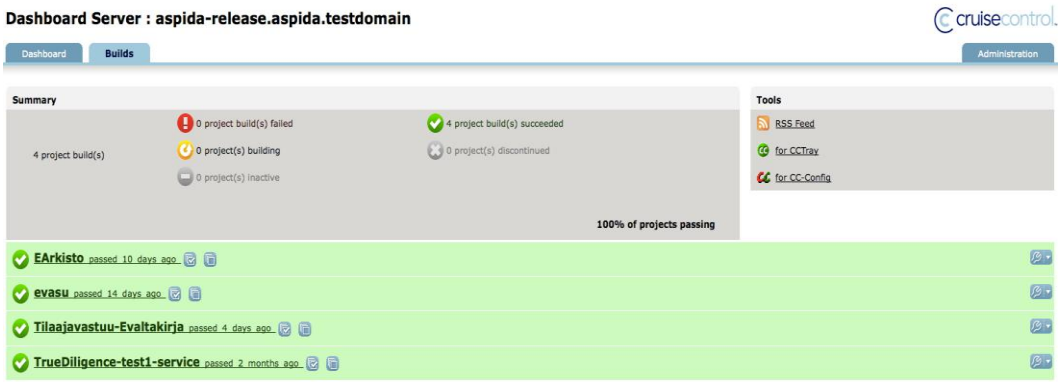
CruiseControlin config.xml koostuu projekti tageistä (<project>), joiden sisään kirjoitetaan halutun projektin tiedot ja toiminnot. CruiseControl-projektit tukevat useita toimintoja, joista yleisimmät ovat Apache Antilla tehdyt projektin tietokantavarastopäivitykset, kokoaminen tai kokoamisen siivous käyttäen build.xml-tiedostoa, kootun paketin kopiointi haluttuun paikkaan ja raportointi kokoamisesta. Asetustiedostolla voidaan ajastaa kokoaminen tapahtumaan haluttuna aikana (pause), aina tietovaraston muututtua (modificationset) tai käyttäjän pyynnöstä (Force build). Perusasetuksilla CruiseControlin toiminnot ovat käytössä Jetty-palvelimen ajamilla sivuilla, joita voidaan muokata halutunlaisiksi tai asettaa käyttäjäoikeuksia (koodi 1).

```
<project name="">
  <property name="" value="" />
   <listeners>
     <currentbuildstatuslistener file="" />
   </listeners>
   <bootstrappers>
     <svnbootstrapper localWorkingCopy="" username="" password="" />
     <antbootstrapper anthome="" buildfile="" target="" />
   </bootstrappers>
   <modificationset quietperiod="">
<svn localWorkingCopy="" RepositoryLocation="" username="" password="" />
```

```
  </modificationset>
  <schedule interval="">
    <ant anthome="" buildfile=""  target=""/>
                    <pause starttime="" endtime="" />
  </schedule>
  <log>
     <merge dir="" />
  </log>
  <publishers>
    <onsuccess>
      <artifactspublisher dest="" File="" />
    </onsuccess>
  </publishers>
</project>
```

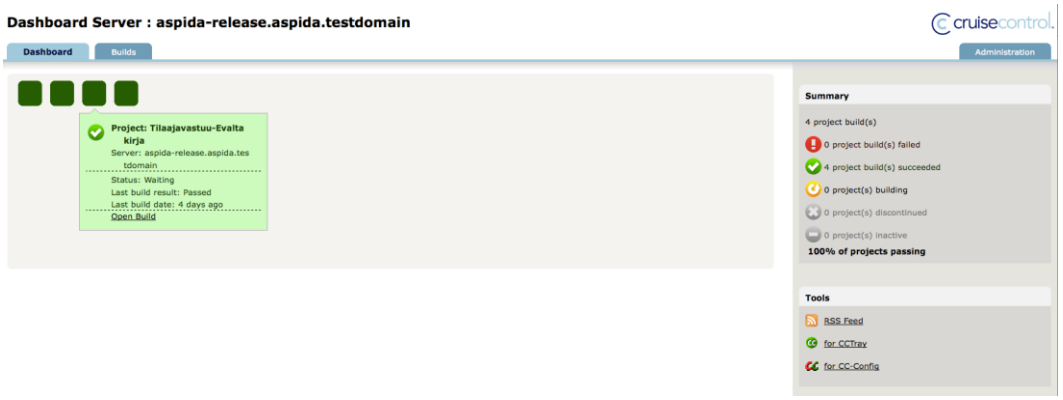
Koodi 1. Jetty-käyttöoikeudet

Käyttöliittymänäkymiä on kolme kappaletta, joista Build (kuva 9) tuntui antavan hyvän kuvan kaikkien projektien tilasta.



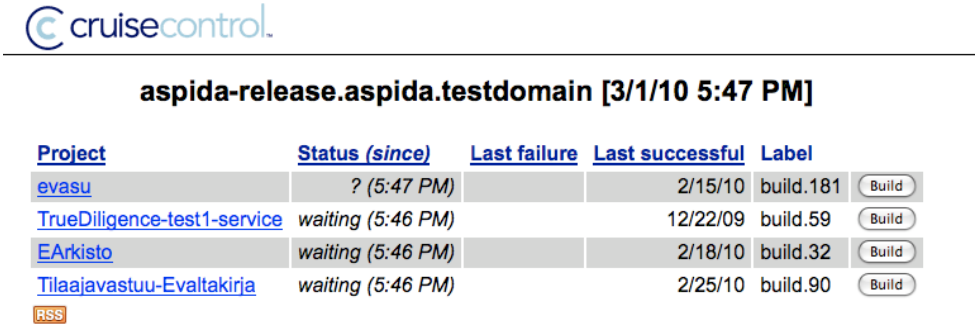
Kuva 9. CruiseControl Builds-välilehti

Näkymästä voidaan nähdä yhdellä silmäyksellä kaikkien koottavien projektien tilat värikoodattuna. Projektin väri on vihreä, jos koonti onnistunut ja punainen jos koonti on epäonnistunut. Kuvassa 10 on toinen perusnäkymä Dashboard.



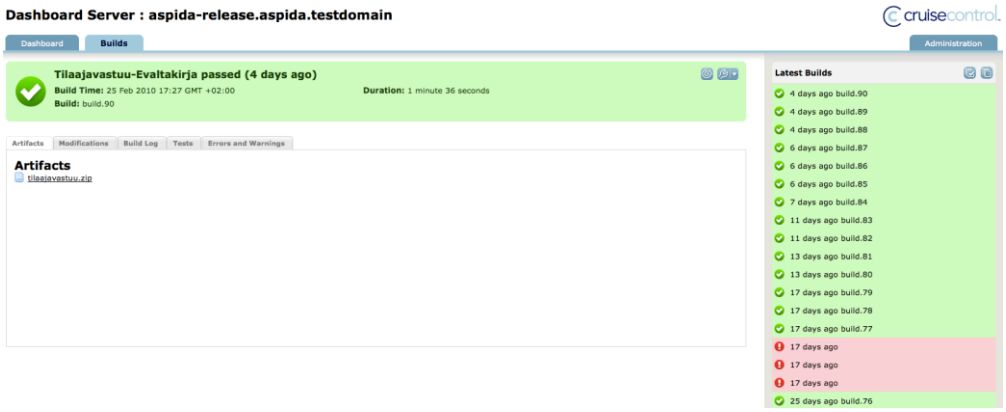
Kuva 10. CruiseControl Dashboard-välilehti

Dashdoard-välilehden näkymässä on esillä samat tiedot kuin Builds näkymässä. Ainoina eroina on, ettei koontiprojektien perustietoja ole suoraan näkyvillä – ainoastaan värikoodaus näkyy ennen hiiren siirtämistä projektin päälle, jolloin tulee esiin projektin tarkemmat tiedot. Kolmantena näkymänä on CruiseControl-perusnäkymä (kuva 11).



Kuva 11. CruiseControl yksinkertainen käyttöliittymä

CruiseController-perusnäkymä on askeettisempi kuin aikaisemmat näkymät, mutta tämä puoli tarjoaa käyttäjälle paremmat mahdollisuudet tarkastella ja muokata CruiseCotrol-projekteja. Koontiprojektin valinnan jälkeen käyttäjä pääsee näkemään tarkempia tietoja projektin koonnista, sekä graafisia esityksiä projektin koontitilasta. Projektin tietoja ja asetuksia pääsee muuttamaan suoraan käyttööliittymästä, jolloin käyttäjän ei tarvitse mennä muokkaamaan näitä tietoja itse palvelimelta. Molemmista näkymistä voidaan kuitenkin hyvin seurata projektien tilaa ja tapahtumia aina kokoamisesta testeihin asti. Parhaimpana näkymänä voidaan pitää kuvan 12 näkymää projektin tilasta ja kasauspaketista.



Kuva 12. CruiseControl projektin tiedot

Näkymästä löytyy myös kohta, josta voidaan tarkistella tehtyjä svn-muutoksia historialistana. Listalta voidaan nähdä kuka muutoksen tehnyt, mihin revisioon se kuuluu ja mihin tiedostoon muutos on tehty. Näkymästä voidaan myös tarkastella koonnin logia, testien tuloksia ja mahdollisia virheitä.

Force build sopii hyvin tuotantoonvientiin tai kokoamisen tekemiseen haluttuna aikana, mutta tuotantoonviennin yhteydessä tämän toiminnon tulisi olla vain tiettyjen tahojen saatavilla. Jos käyttäjille halutaan pääsy selkeään yleiskuvaan Builds-välilehdelle, täytyy force build kytkeä pois päältä vahinkojen välttämiseksi. Kytkentä voidaan tehdä joko kirjoittamalla config.properties- tai dashboard-config.xml-tiedostoon käsky forcebuild=”false”. Dashboard-config.xml-toiminto estää kaikkien projektien pakkokokoamisen. Tämä voidaan kuitenkin tehdä helpommin sallimalla kaikille projekteille omat toimintonsa ja kytkemällä force build pois Builds-näkymästä muokkaamalla latest_builds_profile_partial.vm (koodi 2) ja all.js (koodi 3) -tiedostoja.

```
<a id="${buildCmd.build.projectName}_forcebuild" class="force_build_link"></a>
Muutetaan:
<a id="${buildCmd.build.projectName}_forcebuild" class="force_build_links"></a>
```

Koodi 2. latest_builds_profile_partial.vm-korjaus

```
force_build_links : function(json) {

        },
```

Koodi 3. All.js-korjaus

Tällöin pakkokasaus-toiminto ei ole näkyvissä Builds-sivulla, eikä kukaan pääse vahingossa painamaan kokoamista päälle tuotantoversiolle.

6.1.1 Jetty-käyttöoikeudet

Käyttäjänhallinta mahdollistaa luotettavamman tavan tehdä tuotantoonvientiä, jolloin toimimattomia versioita ohjelmasta ei pääse asiakkaiden käsiin. Käyttäjänhallinnalla voidaan myös määrittää pääsy vain tiettyihin CruiseControl-käyttöliittymän sivuihin ja toimintoihin.

Realm.properties-tiedostoon (koodi 4) asetetaan käyttäjä ja salasana tunnistusta varten.

```
ccadmin: MD5:95f90745bcd0fda1f9ae8c37164c6e93,ccadmin
```

Koodi 4. Realm.properties-asetus

Salasana voidaan kryptata MD5-tarkisteella, jolloin salasana ei ole käyttäjän ymmärrettävässä muodossa, vaan saatu salasana tarkistetaan sitä vasten. Webdefault.xml-tiedostoon (koodi 5) asetetaan käyttäjän tarkistuksen vaativat sivut ja pyyntötavat.

```
<security-constraint>
                <web-resource-collection>
                                <web-resource-name>Default</web-resource-name>
                                <url-pattern>admin/config/*</url-pattern>
                                <url-pattern>tab/build/detail/TrueDiligence/*</url-
                                pattern>
                                <http-method>GET</http-method>
                                <http-method>HEAD</http-method>
                                <http-method>POST</http-method>
            </web-resource-collection>
             <auth-constraint>
                                <role-name>ccadmin</role-name>
             </auth-constraint>
 </security-constraint>
            <login-config>
                                <auth-method>BASIC</auth-method>
                                <realm-name>ccadmin</realm-name>
            </login-config>
            <security-role>
```

```
                    <role-name>ccadmin</role-name>
                </security-role>
```

Koodi 5. Webdefault.xml-lisäys

Lopuksi käyttäjä otetaan käyttöön lisäämällä käyttäjäntiedot Jetty.xml –tiedostoon (koodi 6).

```
<Set name="UserRealms">
 <Array type="org.mortbay.jetty.security.UserRealm">
  <Item>
   <New class="org.mortbay.jetty.security.HashUserRealm">
       <Set name="name">ccadmin</Set>
       <Set name="config"><SystemProperty name="jetty.home"
         default="."/>/etc/realm.properties</Set>
       <Set name="refreshInterval">1</Set>
   </New>
  </Item>
 </Array>
</Set>
```

Koodi 6. Jetty.xml uuden käyttäjän käyttöönottaminen

Esimerkkikoodien perusteella voidaan lisätä useampia käyttäjiä erilaisilla oikeuksilla ja asetuksilla. Eri käyttäjien tekeminen on mielekästä silloin, kun samaa ohjelmaa käyttää useampi ihminen tai ihmisryhmä, joilta halutaan rajata pääsy tiettyihin projekteihin.

6.1.2 Apache Ant

Ant-työkalulla pystytään muun muassa kasaamaan ohjelmistoja halutulla tavalla sekä suorittamaan useita muita toimintoja kuten tiedoston siirtoa verkossa tai käskyjen ajamista palvelimella. Ant mahdollistaa ohjelmiston oman build.xml-tiedoston lisäämisen toiseen build.xml-tiedostoon, jolloin ohjelmiston omaan build.xml ei tarvitse tehdä haluttuja asennuskäskyjä (koodi 7).

```
<trycatch>
        <try>
                <import file="${basedir}/build.xml" />
        </try>
        <catch>
                <echo message="The build.xml file does not exist!" />
        </catch>
</trycatch>
```

Koodi 7. Tiedoston omaisuuksien lisääminen toiseen tiedostoon

Esimerkkikoodi tiedoston lisäämisestä sisältää kokeilu-tagin (trycatch), jolla voidaan kokeilla, onko tiedosto olemassa. Jos tiedoston lisäys onnistuu, niin tiedosto lisätään import tagiin lisätyllä tiedoston osoite ja nimi tiedoilla. Muuten tulostetaan virheilmoitus.

Projekteissa käytettiin normaaleja build.xml-tiedostoja, joihin ei muutoksia tehty. Sen sijaan tehtiin uudet build.xml-tiedostot, joihin ladattiin projektin käyttämä build.xml-tiedosto. Tällöin saatiin tehtyä erillisiä vain integraatioon liittyviä muutoksia ilman, että kehittäjät kärsivät muutoksista. Muutokset koskivat projektin tiedoston muotoa, nimiä, paikkaa ja testipalvelimen asetuksia. Konfiguraatiomuutoksilla saatiin aikaiseksi helposti eri projektien hallinta ja eri asennuskonfiguraatioilla tiedostoiden vaihtaminen projektista toiseen. Eri konfiguraatiotiedostojen asentaminen eri palvelimien web-sovelluksiin helpottaa jokaisen web-sovelluksen testaamista, kun jokaista ohjelmaa ei tarvitse erikseen muokata asennuksen jälkeen. Projektin vieminen ja asentaminen eri palvelimille vaatii muutamia ant-toimintoja kuten ssh- ja scp-yhteyksien käyttämistä ja sshexec-komentoja.

Build.xml-tiedostoon asetetaan muuttujia, jotta toimintojen käyttäminen muissa projekteissa onnistuu helposti ilman ongelmia (koodi 8).

```
<property name="service" value="service"/>
```

Koodi 8. Muuttujan lisääminen

Uusien kohteiden ja niiden riippuvuuksien lisääminen tapahtuu koodin 9 mukaisesti.

```
<target name="build" depends="war">
```

Koodi 9. Kohteiden riippuvuuden lisääminen

Kohteita voidaan lisätä eri toimintoja varten lisäämällä name kohtaan tieto kohteen nimestä. Kohteiden riippuvuuksia on helppo muuttaa depends-tiedolla, jolloin turhat toiminnot voidaan tarvittaessa kytkeä pois päältä vain poistamalla riippuvuus.

SSH-yhteydellä voidaan toteutetaan palvelimella tarvittavat käskyt, jolloin käyttäjän ei tarvitse myöhemmin suorittaa näitä käskyjä itse. Tällaisia käskyjä voivat olla palvelimen palveluiden (services) esimerkiksi httpd:n sammuttaminen tai käynnistäminen, tiedostojen purku tai kopiointi ja erilaisten shell-skriptien ajaminen (koodi 10).

```
<sshexec host="${address}" username="${user}" keyfile="${key.file}" verbose="true"
trust="true" passphrase="" command="/aspida/webapps/${shfilename}" />
```

Koodi 10. Komennon suorittaminen Linux virtuaalikoneella

SSH-käskyssä kerrotaan etäpalvelimen osoite (host), käyttäjätunnus (username), avain-tiedoston (keyfile) osoite sisäänkirjautumista varten ja suoritettava käsky (command). Tiedostojen siirto koneelta toiselle toteutetaan Apache Antin scp -komennolla (koodi 11).

```
<scp file="C:/Program Files/CruiseControl/config-
build/config/test1/${service}/PureDBUpgrader.xml" to-
dir="${user}@${address}:/aspida/webapps/${service}/WEB-INF/classes/configs" key-
file="${key.file}" verbose="true" trust="true" passphrase="" />
```

Koodi 11. Tiedoston siirto

SCP komento sisältää siirrettävän tiedoston (file) osoitteen, siirrettävän tiedoston siirtopaikan (todir) ja avaintiedoston (keyfile) osoitteen sisäänkirjautumista varten.

Selenium-testien ajo Windows-koneelle voidaan toteuttaa monella tapaa, joista helpoiten ylläpidettävä tapa on käyttää bat-päätteistä tiedostoa. Kun testien ajokoodi on eriytetty muusta koodista, voidaan sitä muuttaa myöhemmin tarpeen mukaan ilman, että se vaikuttaisi muuhun ohjelman

toimintaan. Apache Antilla Windows-koneella oleva bat-tiedosto ajetaan koodin 12 mukaisesti.

```
<target name="seleniumtest">
                <exec dir="${selenium}" executable="cmd" os="Windows XP">
                                <arg value="/c"/>
                                <arg value="LoginLogout.bat"/>
                                <arg value="-p"/>
                </exec>
</target>
```

Koodi 12. Komennon suorittaminen Windwos-virtuaalikoneella

Käynnistyskoodi on eritetty omaksi kohteeksi, ja se sisältää exec-tagin, jossa on kohteen osoite (dir), käyttäjärjestelmän tiedot (os) ja ajettavan tiedoston nimen (value).

6.2 Shell-skriptit

Shell-skripteillä voidaan toteuttaa asennus, asennuksen siistiminen, varmuuskopiotiedostojen teko tietokannasta ja eri palveluiden käynnistäminen. Sillä voidaan ohjata kaikkia asennuksen toimintoja aina palvelimen sammuttamisesta sen käynnistämiseen asti.

Kaikki Linux-asennukset on toteutettu käyttäen bash-skriptejä, joissa tehdään varmuuskopiointi palvelusta ja tietokannasta, palveluiden siistiminen ja tarvittavat tietokantamuutokset ennen palvelun purkua. Ennen palvelun varmuuskopiointia (koodi 13) ja tyhjennystä tarkistetaan, onko palvelun tiedosto jo poistettu:

```
# Backup service if files exist

if [ -e /aspida/webapps/$service/"Release Notes.html" ];

then echo "File Release Notes.html exist, making backup!"

cd /aspida/webapps/

zip -r $service.zip $service
```

Koodi 13. Varmuuskopiointi

Poistossa (koodi 14) kannattaa käyttää komentoja, joissa poistokomento suoritetaan joissain muualla kuin juuressa, sillä jos poistossa on alussa kenomerkin jälkeen välilyönti, tyhjentyy koko kovalevy, eivätkä vain halutut tiedostot.

```
# Service command

cd /aspida/webapps/

sleep 2

rm -rf $service/*

sleep 2
```

Koodi 14. Tiedostojen poisto

Kaikkien vaiheiden välissä kannattaa muistaa laittaa tauko, jotta komento ehditään tehdä loppuun ennen seuraavan aloittamista.

Mysql-taulun solun poisto (koodi 15) suoritetaan execute-käskyllä.

```
#mysql --execute="DELETE FROM pure_version_repository.VersionData WHERE cus-
tomerName='$text_$s'";
```

Koodi 15. Mysql-taulun solun poisto

Palautus voidaan tehdä käyttäen Putty ssh -ohjelmaa tai Linuxin terminaalia. Ensin kannattaa sammuttaa (koodi 16) ja tyhjentää palvelu ennen varmuuskopion palautusta. Palautus voidaan tehdä suoraan purkamalla tehdyt tiedostot palveluun ja käynnistää palvelu sen jälkeen.

```
# Tomcat 6 stop

service tomcat6 stop

# Tomcat 6 start

service tomcat6 start
```

Koodi 16. Tomcatin pysäytys ja käynnistys

Tuotantoon tehtiin omanlaisensa shell skripti -tiedostot, jossa käyttäjältä vaaditaan vähän enemmän osallistumista, sillä tuotannossa joudutaan tekemään tietty versio palvelusta aktiiviseksi (koodi 17).

```
#!/bin/sh
mysql --execute="SELECT schemaName FROM pure_version_repository.VersionData
 ORDER BY schemaName";
read -p "Give new database version number [2_4_0]: " newd
read -p "Give old database version number [2_3_0]: " oldd
echo Pure Version rows to delete:
echo truediligence_"$newd"
echo truediligence_"$newd"_s2
echo
echo Pure Version rows to update:
echo truediligence_"$oldd"
echo truediligence_"$oldd"_s2
while true; do
 read -p "Are you sure to continue? yes[y] or no[n]: " answer
    case $answer in
       [Yy]* ) break;;
       [Nn]* ) exit;;
       * ) echo "Please answer yes[y] or no[n]: ";;
    esac
done
# Database command
#Delete
echo Deleting pure version truediligence_"$newd" rows
mysql --execute="DELETE FROM pure_version_repository.VersionData WHERE sche-
maName='truediligence_"$newd"'"; > 2
sleep 2
#Drop
echo Dropping truediligence_"$newd" database
mysql --execute="DROP DATABASE truediligence_"$newd"";
sleep 2
#Update
echo Updating pure version truediligence_"$oldd" row
mysql --execute="UPDATE pure_version_repository.VersionData SET active=1 WHERE
schemaName='truediligence_"$oldd"'";
 exit 0
```

Koodi 17. Version palautus

Käyttäjälle haetaan tietokannasta kaikki versiot, joita tietokannassa on. Tämän jälkeen skripti määrittelee, mikä on ohjelman uusin ja toisiksi uusin versio. Ennen varsinaista palautusta toimintokäyttäjältä varmistetaan vielä, että ohjelma on tekemässä oikeat muutokset. Käyttäjälle on näkyvissä tulostus muutettavista ja poistettavista tiedoista. Tämän jälkeen tietoja käytetään yleisinä muuttujina käskyissä. Skriptin ajatuksena oli tehdä jokaisesta käyttäjän vaiheesta mahdollisimman helppo ja vähäriskinen.

6.3 Selenium

Selenium on testaukseen tarkoitettujen ohjelmistojen tuoteperhe, joka sisältää muun muassa Selenium Core-, Selenium IDE- ja Selenium Remote Control -ohjelmat.

Selenium Core on kaikkien testien ajon ydin, joka perustuu Javascript-ohjelmointikieleen. Ydintä voidaan käyttää itsenäisenä Javascript- tai HTML-kielisten testien ajossa. Ydin kuuluu Selenium Remote Control -pakettiin, joka mahdollistaa muiden ohjelmointikielten käytön.

Selenium IDE -ohjelmalla voi tehdä perustestejä, joita sitten käytetään muiden testien pohjana kopioimalla tuotettu koodi haluttuun tiedostoon. Selenium IDE:ssä on helppo ja yksinkertainen käyttöliittymä ja se on saatavilla Firefox-lisäosana (plugin). Ohjelma nauhoittaa käyttäjän selaimessa tekemät toiminnot muistiin, joita sitten voidaan tarvittaessa muuttaa samalla tai jälkeenpäin. Ohjelma tarjoaa myös testien ajomahdollisuuden, jossa testit ajetaan suoraan avonaisessa selaimessa. Ohjelman käyttäminen onnistuu ilman mitään ohjelmointikokemusta.

Selenium Remote Control on palvelin, joka ohjaa ja ajaa testejä, jotka voidaan syöttää eri ohjelmointikielillä kuten Javalla ja HTML:llä. Syötetyt testit muunnetaan Seleniumin omaan käskymuotoon, ja käskyjen vaikutuksen näkee suoraan avautuneessa selaimessa.

Seleniumin tarjoamat HTML-testit ovat peruskäytössä hyviä ja nopeita tehdä ja toteuttaa. HTML-testien suurimpana ongelma on niiden riittämättömyys vaikeampiin ja monimutkaisempiin testeihin, joissa yritetään testata mutkikkaampia ja vaikeimpia asioita kuin vain palvelun painikkeita ja tekstikenttiä. Ongelmia tulee elementtien tunnistuksien kanssa, eikä modulaarinen rakenne onnistu samoin kuin Javalla. Tunnistusongelmat

voidaan ratkaista Xpath-navigointia apuna käyttäen. Xpath auttaa tarkentamaan kohteen palvelusta, jos id-tunnistusta ei voida tehdä tai se ei onnistu.

Javalla pystytään tekemään erittäin modulaarisia testejä ja niiden ansioista testien päivityksistä tulee helppoja ja nopeita. Kaikki testit voidaan jakaa omiin luokkiinsa ja niitä voidaan ajaa esimerkiksi Java-listojen avulla. Alussa ongelmana oli virheen paikannus, sillä kun virhe sattui, se lopetti koko testin. Ajolistojen ansiosta voidaan ajaa useita testejä ilman, että testien ajo loppuu ensimmäiseen virheeseen.

Seleniumilla toteutettiin web-palvelujen toimimisen testaus haluttuna aikana. Tämä voidaan tehdä aikatauluttamalla testit Windows Scheduled Task -toiminnolla, jolla käynnistetään testit bat-tiedostoa apuna käyttäen. Testit voidaan ajaa haluttuna aikana tai ajanjaksona ja testien tuloksia voidaan tarkkailla testien tulostiedostosta. Tulosten automaattinen tarkastus toteutettiin käyttäen Linux cron –toimintoa, jolla lähetetään virhetilanteissa ilmoitus asiasta vastaavalle henkilölle.

7 TESTIEN TEKEMINEN JA AJO

Seleniumin asennuksen jälkeen testejä ryhdyttiin tekemään html-typpisinä testeinä. HTML-testit voidaan tehdä suoraan nauhoittamalla halutut tapahtumat Firefox-selaimella käyttäen Selenium IDE -ohjelmaa. Testien tekeminen html-tyyppisesti tapahtuu erittäin helposti ja niitä voidaankin testata heti nauhoituksen jälkeen ennen tallentamista. Testit nauhoitetaan samalla, kun käyttäjä käyttää haluttua web-sovellusta. Testi voidaan ottaa käyttöön heti tallentamisen jälkeen tekemällä sille suite.html- (liite 3) ja result.html-tiedostot (liite 4). Suite pitää sisällään tiedon ajettavista testitiedostoista ja result -tiedostoon tulee testin tekemät tulostiedot. Mainittujen tiedostojen käyttäminen eli käynnistäminen tapahtuu käyttämällä bat- tai shell script -tiedostolla, jossa käynnistetään Selenium palvelin ja annetaan sille tiedot halutuista toiminnoista ja kohteista. Käynnistettävän tiedoston tyyppi bat tai shell script riippuu käytettävästä käyttöjärjestelmästä. Jos testit suoritetaan Windows-koneella, joudutaan käyttämään bat-tyyppistä tiedostoa, Linux-puolella käytetään shell script -tiedostoa. Käynnistyksessä käytettävä koodi 18 on esimerkki Windows-puolen käynnistystavasta.

```
java -jar selenium-server.jar -htmlSuite "*chrome" "URL"
"c:\absolute\path\to\HTMLSuite.html" c:\absolute\path\to\results.html"
```

Koodi 18. Testin käynnistys [5.]

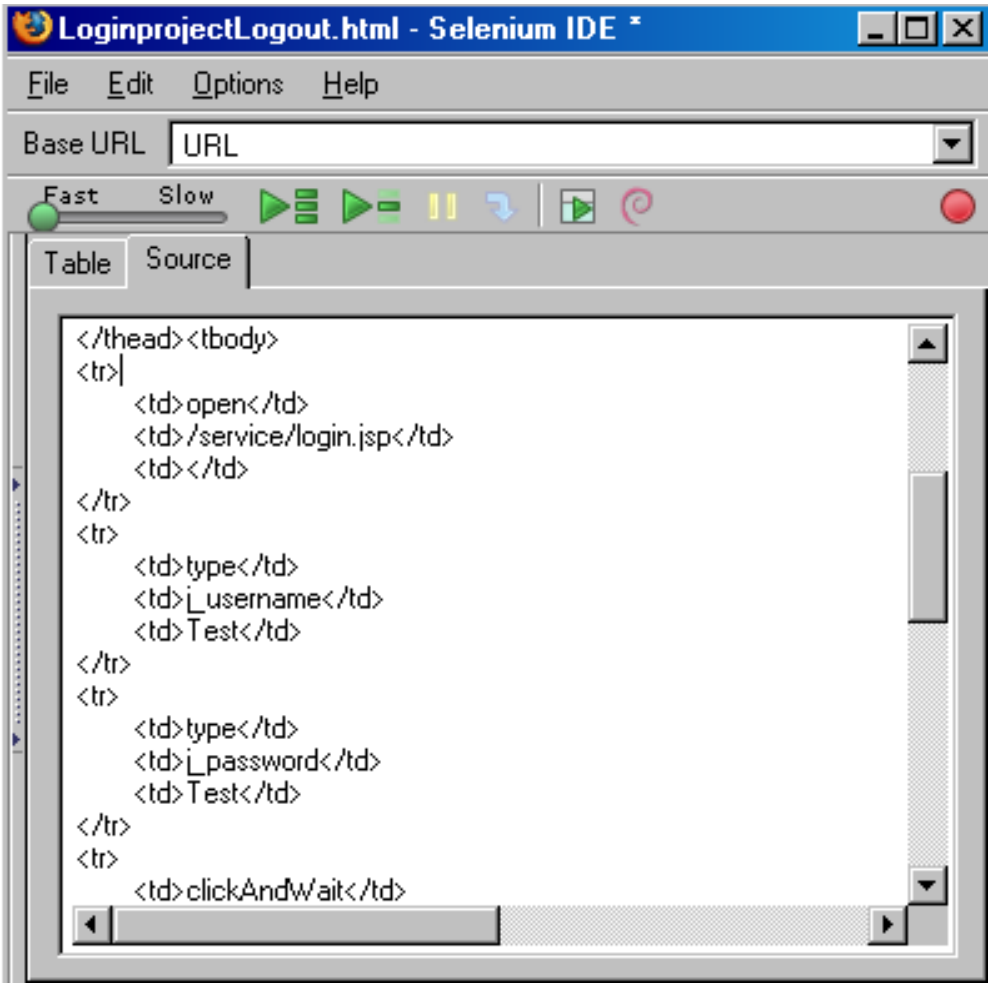
Selenium-palvelin käynnistetään Java-käskyllä, ja htmlsuite kertoo, että testissä käytetään suite-tiedostoa. Tämän jälkeen esitellään käytettävä selain, aloitusosoite ja tiedostojen sijainti. Jos käyttäjä haluaa ja yleensä joutuukin käyttämään valmista IE- tai Firefox–profiilia, joutuu käynnistykseen lisäämään tiedon, mistä profiilin löytää (koodi 19).

```
-firefoxProfileTemplate "c:\absolute\path\to\firefoxprofile"
```

Koodi 19 . Firefox-profiilin lisääminen [5.]

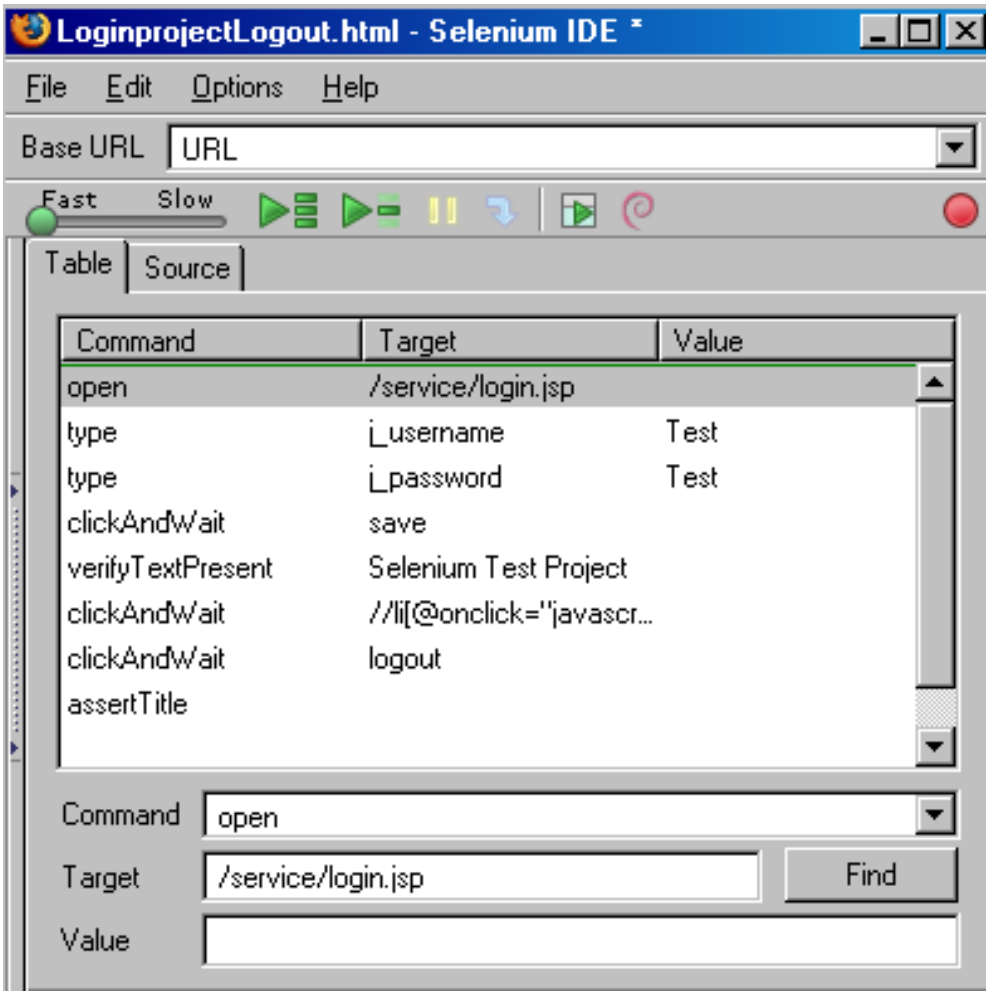
Selenium-palvelimen voi myös käynnistää ilman näitä kaikkia parametreja. Ainoa parametri, joka tulee asettaa, on interactive-tilaparametri, jolloin ohjelma jää päälle odottaen ajettavia testejä tai käyttäjän antamia komentoja.

Selenium IDE:ä käyttäessä kannattaa kuitenkin ottaa huomioon, että se on työkalu, jolla tehdään pohjatyö nauhoittamalla halutut toiminnat suoraan selainta käyttäen, eivätkä sillä tehdyt testit toimi täysin varmasti. Ongelmatilanteita esiintyy yleensä sivujen latausajoissa, painikkeiden käyttämisissä ja yleisessä ohjelman hitaudessa. Pelkkä käyttäjän toimintojen nauhoittaminen ei siis riitä, vaan testejä kannattaa muokata käsin, jotta ne toimisivat ajettaessa varmemmin. Muokkaus voidaan tehdä joko testien nauhoitusvaiheessa tai nauhoituksen loputtua muokkaamalla saatua koodia (kuva 13).



Kuva 13. Selenium IDE -nauhoituskoodi

Muokkausvaiheessa voidaan joko lisätä komentoja jo olevien komentojen väliin tai muokata annettuja käskyjä ja kohteita. Muokkaus voidaan tehdä suoraan HTM-koodiin tai käyttää komentolistausta taulukkopuolella (kuva 14).



Kuva 14. Selenium IDE -nauhoituskäskyt

Testit voivat olla HTML-muotoisena sivuna (liite 1) tai Java-tyyppisenä koodina (liite 2). Molemmille yhteistä on käskyjen, kuten paina (Click) antaminen itse testiä suorittavalle ohjelmalle. Liitetiedostoissa 1 ja 2 on esitetty karsittu esimerkki Tilaajavastuu.fi-sivujen testaamisesta. Sivuilta voidaan testata sivun kuvia tai muita elementtejä joko id -tunnisteella tai Xpath-osoitteella [4]. Tallennetut testit voidaan ajaa Selenium Remote Controlilla käyttämällä joko HTML- tai Java-rajapintoja. Java-testit voi ajaa käynnissä olevaan Selenium-palvelimeen kääntämällä Java-tiedosto class-tiedostoksi javac-komennolla (koodi 20).

```
javac -classpath C:\Selenium\selenium-remote-control-1.0-beta-2\selenium-java-client-driver-1.0-beta-2\selenium-java-client-driver.jar tilaajavastuu.java
```

Koodi 20. Java–tiedoston kääntäminen [5.]

Käännetty tiedosto ajetaan Java–komennolla (koodi 21).

```
java -classpath C:\Selenium\selenium-java-client-driver.jar;. tilaajavastuu
```

Koodi 21. Kännetyn tiedoston ajaminen [5.]

Testi itsessään ei tee luettavaa raporttia testien tuloksista, vaan se tulostaa testien onnistumisen komentokehotteeseen. Raportointi saadaan toimimaan logging-selenium.jar-tiedostolla, joka vaatii koodin lisäämistä testiin. Muutoksia täytyy tehdä setup-metodiin ja koodin loppuun tulee lisätä tearDown-metodi (liite 5). Lisättävällä koodilla mahdollistetaan kunnollinen raportointi ja kuvakaappauksien lisääminen raporttiin. Raportoinnin lisääminen vaatii jar-tiedostopolun lisäämistä käynnistyskomentoon.

Jos ohjelmassa käytetään lisäpakettikirjastoja, tiedoston kääntäminen vaatii lisäyksiä kääntökäskyhin. (koodi 22).

```
echo Käännetään testiä

javac -classpath .;C:\selenium-remote-control\selenium-java-client-driver\selenium-
java-client-driver.jar;C:\selenium-remote-control\selenium-java-client-driver\junit-
4.5.jar;C:\selenium-remote-control\selenium-java-client-driver\logging-selenium-
1.2.jar;C:\\selenium-remote-control\selenium-java-client-driver\commons-lang-2.4.jar
JavaLoginLogout.java

echo Testien ajo
java -classpath .;C:\selenium-remote-control\selenium-java-client-driver\selenium-java-
client-driver.jar;C:\selenium-remote-control\selenium-java-client-driver\junit
-4.5.jar;C:\selenium-remote-control\selenium-java-client-driver\logging-selenium-
1.2.jar;C:\selenium-remote-control\selenium-java-client\commons-lang-2.4.jar.
 JavaLoginLogout

echo Testit suoritettu
```

Koodi 22 . Java-testien kääntäminen lisäpakettien kanssa

Kääntämisen yhteydessä pitää määrittää, mitä lisäpaketteja on käytössä tai luokan kääntäminen ei onnistu eikä lisäominaisuuksia saada käyttöön.

7.1 Selenium Java

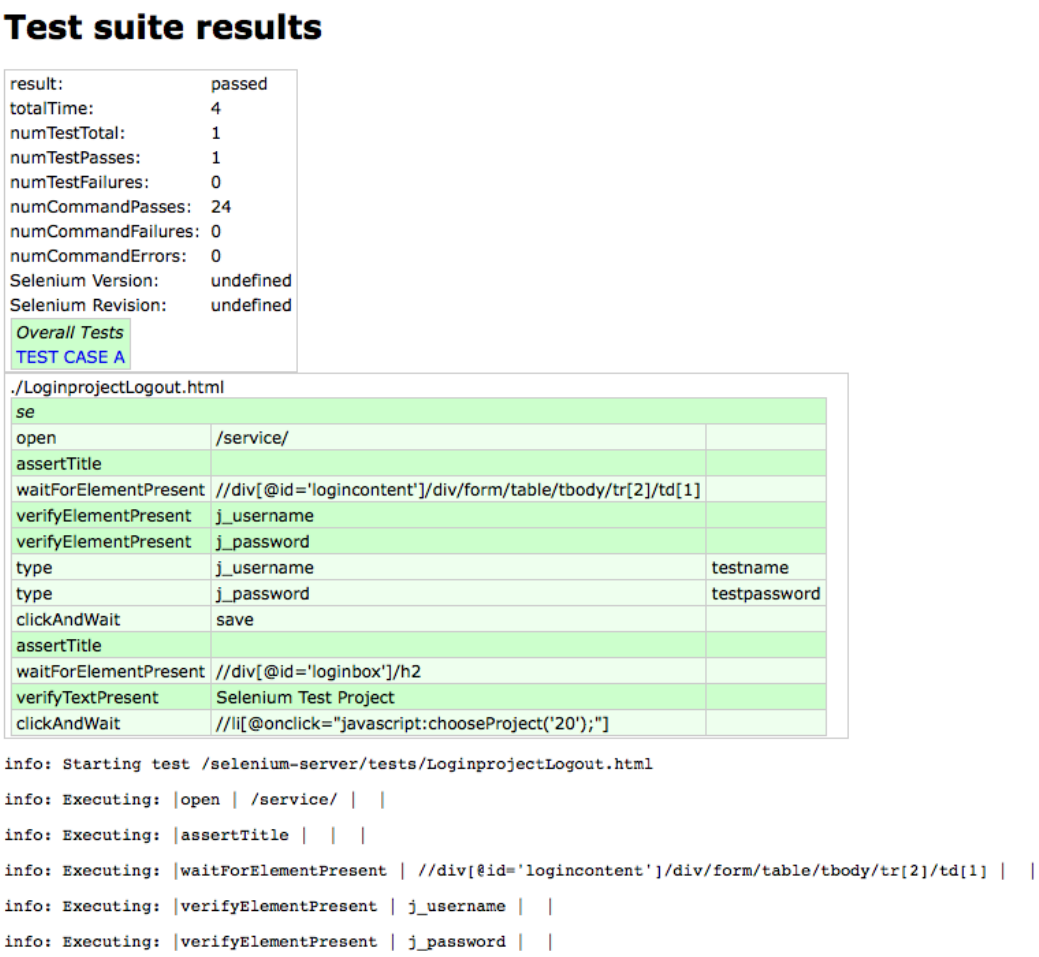
Selenium IDE:n HTML -tyyppisillä testeillä ei päästä kovinkaan pitkälle, jos halutaan testata ohjelmaa mahdollisimman monipuolisesti ja laajasti. HTML-testit toimivat loistavasti pienissä ja yksinkertaisissa testeissä, joilla on tarkoitus testata ohjelman sivujen latautuminen ja niiden olemassaolo. HTML-testit ovat helppoja ja nopeita tehdä ja niiden toteuttaminen onnistuu vaikka suoraan selaimesta. Parempaan ja laajempaan testaamiseen voi käyttää esimerkiksi Seleniumin tarjoamaa Java-tukea, jolla pystytään toteuttamaan suuria testikokonaisuuksia järjestelmällisesti. Java-tuki tarjoaa yksinkertaisten komentojen syöttörajapinnan, jolla pystytään ohjaamaan testejä.

7.2 Java-testit

Perus Selenium Java-testien ajoluokka sisältää vähintään setUp-, testNew- ja tearDown-metodit, joilla pystytään suorittamaan testejä. Luokka vaatii aina mukaansa SeneseTestCase-luokan lisäyksen (extends). Testien ajo alkaa asetusten asettamisesta setUp-metodissa. Testien suorittaminen tapahtuu testNew metodissa ja testien lopetus tearDown-metodin käskyillä. SetUp-metodille tarvitaan komento selenium.start, jolla palvelu käynnistetään. Käynnistystä ennen pitää asettaa palvelimen tiedot, portti, aloituskäsky ja selainosoite. Komento avaa uuden selaimen, jossa testit ajetaan. TearDown-kohdassa yleensä vain suljetaan palvelu komennolla selenium.stop. Tällöin testejä varten avattu selain sulkeutuu. TestNew-metodin sisällä ajetaan valitut testit/komennot, jotka toteutetaan Selenium-palvelimessa, ja toteutus näkyy selaimessa.

Testauksen keskeytysongelma voidaan ratkaista jakamalla ajettavat testit osiin ja ajamalla ne erillisinä osina. Esimerkiksi sisäänkirjautumisesta voidaan tehdä oma yksittäinen testi eli komentosarja, joka voidaan ajaa ennen jokaista testiä, jossa vaaditaan sisäänkirjautuminen. Parhaaksi menettelyksi on osoittautunut menetelmä, jossa kaikki testattavat sivut jaetaan omiin luokkiinsa, joihin tehdään omat metodit ja metodeille omat komennot. Tällä tavoin voidaan jokaisen sivun toiminnot jakaa omiksi kokonaisuuksiksi esimerkiksi painikkeille, filtteröinneille ja syötekentille. Tällöin jos sivu muuttuu osittain tai kokonaan, ei tarvitse muuta kuin muuttaa luokan metodien komennot toimiviksi uutta testiä varten. Näin ollen jokaisesta testiluokasta voidaan tehdä oma testilista, joka voidaan ajaa

täysin omana kokonaisuutenaan. Tämä mahdollistaa erittäin monipuolisten testien suunnittelun ja niiden helpon toteutuksen. Kun kaikki testit pystytään räätälöimään halutulla tavalla ja valitsemalla halutut toiminnot/metodit testilistaan, pystytään selvästi erittelemään jokaisen sivun ominaisuudet omiin osioihinsa. Testien osien jakaminen yksittäisiin osiin mahdollistaa testien koostamisen useammasta testilistasta, jotka voidaan sitten ajaa saadun listauksen järjestyksessä. Näin yksittäinen virhe ei lopeta kaikkien testien ajoa, vaan mahdollistaa useamman testin toteuttamisen ennen lopullista raporttia. Tällöin raportista saadaan mahdollisimman kattava ja laaja, jossa voi esiintyä useita eri virheitä tai muita ongelmia. Javalla toteutetuissa testeissä voidaan myös ottaa halutuista kohdista kuvat, jotka pystytään liittämään lopulliseen raporttiin. Kuvat tarjoavat käyttäjille lisätietoa testien onnistumisesta, näkymien tilasta tai ohjelman ongelmista. Kuvien kaappaus mahdollistaa myös eri selaintyyppien, kuten Internet Explorer 6, 7 ja 8 ja FireFox 2 ja 3 aiheuttamien ongelmien vertaamisen haluttuun ulkoasuun. Testien ajon välissä voidaan myös lisätä raporttiin kommentteja, jotka helpottavat raportin lukemista ja ongelmatilanteiden selvittämistä. Raporteja on vaikea lukea niiden pitkän pituuden ja täynnä käkyjä olevan sisällön takia. Pelkät käskyt eivät välttämättä kerro lukijalle missä kohtaan testissä ollaan ja mitä testiä on juuri suoritettu (kuva 15).



Test suite results

result:	passed
totalTime:	4
numTestTotal:	1
numTestPasses:	1
numTestFailures:	0
numCommandPasses:	24
numCommandFailures:	0
numCommandErrors:	0
Selenium Version:	undefined
Selenium Revision:	undefined

Overall Tests
TEST CASE A

./LoginprojectLogout.html

se		
open	/service/	
assertTitle		
waitForElementPresent	//div[@id='logincontent']/div/form/table/tbody/tr[2]/td[1]	
verifyElementPresent	j_username	
verifyElementPresent	j_password	
type	j_username	testname
type	j_password	testpassword
clickAndWait	save	
assertTitle		
waitForElementPresent	//div[@id='loginbox']/h2	
verifyTextPresent	Selenium Test Project	
clickAndWait	//li[@onclick="javascript:chooseProject('20');"]	

```
info: Starting test /selenium-server/tests/LoginprojectLogout.html
info: Executing: |open | /service/ |  |
info: Executing: |assertTitle |  |  |
info: Executing: |waitForElementPresent | //div[@id='logincontent']/div/form/table/tbody/tr[2]/td[1] |  |
info: Executing: |verifyElementPresent | j_username |  |
info: Executing: |verifyElementPresent | j_password |  |
```

Kuva 15. Esimerkki testin tuottamasta raportista

Testiraporttiin voidaan lisätä kommentteja tuloksien lukemista helpottamaan (koodi 23).

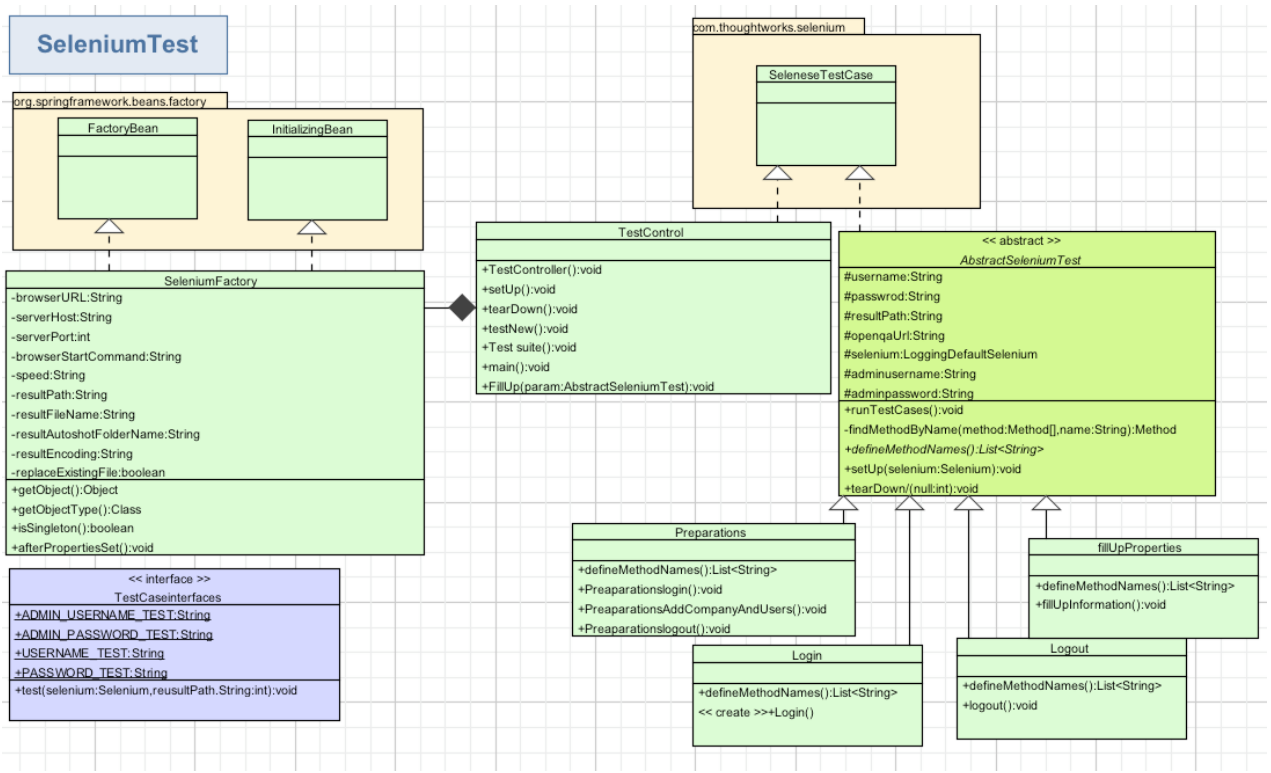
```
selenium.logComment("Comment");
```

Koodi 23. Kommentin lisääminen raporttiin [5.]

Javalla toteutettua testausta voidaan parantaa vielä lisäämällä Spring-hallintasysteemi mukaan testausohjelmaan. Spring-toiminnallisuudella voidaan hallita paremmin luokkia ja käskyjen syötöissä tarvittavia parametreja. Spring tarjoaakin loistavan lisän muuttujien syöttöön ja ominaisuuksien jakamiseen luokkien välillä, jolloin kaikkea samaa koodia ei tarvitse kirjoittaa jatkuvasti uudelleen. Esimerkiksi Selenium-käskyjen lähettäminen voidaan tehdä vain käskyllä selenium, jonka toteutus voidaan määrittää vaikka abstraktissa luokassa. Samaa toteutustyyppiä voi käyttää hyväksi testilistojen ajon yhteydessä.

SeleniumTest-ohjelman toteutuksessa on käytetty Springiä ominaisuuksia apuna, kuten ApplicationContext.xml-luokkien sitomista varten sekä parametrien välittämistä varten. Ohjelman tarvitsemat asetukset on asetettu Config.properties-tiedostossa.

TestController-luokka toimii testien käynnistysluokkana ja ohjaavana ohjelman suoritetusta ohjaavana luokkana. AbstracSeleniumTest-luokalla hallitaan suoritettavia testilistoja ja sillä asetetaan ohjelman tarvitsemat parametrit. SeleniumFactory-luokalla asetetaan Selenium asetukset, kuten kuvatiedostojen tallennuspaikka ja raporttitiedoston paikka. Koko projektin luokkarakenne on esitetty UML-kaaviossa kuvassa 16.



Kuva 16 . Selenium-projektin UML-kaavio

Testien ajaminen perusasetuksilla onnistuu yleensä ilman ongelmia, mutta ongelma minimiasetus-typpisessä testauksessa (liite 5) on sen virheherkkyys, sillä testaus loppuu heti ensimmäiseen virheeseen ja testatuksi tulevat vain siihen asti ajetut testit. Tällöin testaus voi loppua heti alettuaan tai puolessa välissä testejä, jolloin testejä on ehditty ajaa vasta muutama. Tämän tyyppinen testaus tuottaa yleisesti vain yhden virheen kerralla eikä tarjoa ohjelman laajempaan testaamista ennen havaitun virheen korjaamista.

7.3 Raportointi ja kuvankaappaus

Selenium-raportointi kannattaa hoitaa käyttämällä SeleniumLogging-lisäkirjastoa. Lisäkirjasto on tarkoitettu selenium-remote-control-paketin lisäosaksi. Sillä voi tehdä raportin testien ajonaikana ja se mahdollistaa muun muassa ajan ottamisen ja kuvankaappausten lisäämisen raporttiin. Lisäkirjastolla testien tulokset tulevat automaattisesti HTML-muotoon. Lisäkirjaston toiminta vaatii toimiakseen common-lang-kirjaston lisäämisen projektiin. [6.]

Kuvankaappaus on hyvä tapa saada eri testaustilanteista kuva, joka selventää tapahtumia. Kaappausta voidaan käyttää esimerkiksi eri syötekenttien kuvaamiseen ennen ja jälkeen testien, jolloin voidaan helposti validoida käyttöliittymän oikea toiminta. Kuva kaapataan selenium-käskyllä captureScreenshot (koodi 24), joka ottaa tilannekuvan ajon selainikkunasta.

```
selenium.captureScreenshot(resultPath + "screenshots/LoginScreen.png");
```

Koodi 24. Kuvankaappausten ottaminen [5.]

Kuvankaappaus tuo erittäin hyvän ja käytännöllisen lisän virhetilanteiden selvittämiseen ja niiden toistamiseen käsin, kun halutaan tarkempaa tietoa ongelmatilanteista ja niiden syistä.

7.4 Testit ja Xpath

Xpath-toimintoa voidaan käyttää, kun sivun elementti esimerkiksi tekstikenttä on vaikeasti löydettävissä, eikä sitä pystytä löytämään normaalilla tunnistuksella. Xpath on tarkoitettu XML-dokumenttien elementtien ja attribuuttien löytämiseen, mutta sitä voidaan myös käyttää apuna HTML-sivujen elementtien kohdentamisessa. [7.]

Firefox tarjoaa Xpath-lisäosan, jolla pystyy tunnistamaan eri sivun elementien tarkan paikan sivulta [8]. Lisäosan antama osoite voidaan kopioida suoraan selenium käskyyn (koodi 25), jolloin käskyn toiminto varmasti löytää kohteensa.

```
selenium.click("xpath=//html/body/div[@id='wrapper']/
div[@id='content']/div[2]/div[4]/table/tbody/tr[1]/td/a");
```

Koodi 25. HTML-sivun elementin osoitteen lisääminen

7.5 Testidatan luominen Selenium-testillä

Selenium-testillä voidaan luoda myös suuria määriä testidataa tietokantaan ilman, että kaikki pitäisi olla valmiina kannassa. Tiedot kantaan voidaan luoda perinteisellä for-loopilla, jolla saadaan haluttu määrä dataa, jota voidaan sitten hyväksikäyttää testeissä. Tämä sopii parhaiten, kun halutaan luoda testeille sopivat puitteet eli asettaa testeille tarvittavat tiedot niiden toteuttamista varten. Parhaiten tämä onnistuu, kun tarvittavat komennot lisätään omaan luokkaansa, joka ajetaan ennen kaikkia testejä. Tietokannan täyttämistä voidaan myös käyttää ennen käsin tehtäviä testejä, jolloin perusvalmistelut voidaan unohtaa. Normaalisti tähän käytettäisiin tietokantakopiota, joka ajettaisiin suoraan kantaan. Tällöin käyttöliittymän tiedonlisäystoiminnot jäisivät testaamatta, jolloin niitä ei tulisi aina testattua.

Yhtenä vaihtoehtona on testata palvelimen kuormitusta suurilla määrillä dataa, jota syötetään nopeassa tahdissa. Kuormitus voidaan tehdä usealla Selenium-palvelimella. Tällöin voidaan helposti varmistaa ohjelman oikea toiminta suurien kuormien varalta.

8 MAHDOLLISUUDET JA PARANNUSAJATUKSET

Testausta voidaan parantaa ja helpottaa toteuttamalla testit modulaarisesti omina luokkinaan ja metodeinaan. Testausta voidaan parantaa graafisella käyttöliittymällä, joka toimii testauskeskuksena. Testaus voidaan toteuttaa useammalla eri palvelimella ja palvelinta voidaan vaihtaa tarpeen mukaan Selenium Grip-ohjelmalla, joka etsii automaattisesti vapaan palvelimen testejä varten. Käyttöliittymäversioon voi lisätä tietokannan, johon voidaan tallentaa kaikki halutut toiminnot eli komennot. Komentoja tallentaessa ne voidaan tallentaa tiettyyn ryhmään ja samalla voidaan tallentaa ryhmälle oma kuva testattavasta sivusta. Tällöin joka toiminnolle tai nappulalle tulisi oma komento, joka voidaan antaa millä tuetulla kielellä tahansa, sillä ohjelma voi kääntää komennot aina Selenium-käskyksi ennen ajoa. Testien toteutusta ja suunnittelua voidaan helpottaa käyttöliittymälläkuvilla, joita vasten testaaja voi suunnitella uusia testejä näkemällä testin tilan. Samalla testaaja voi päättää syötteen tiettyyn kenttään ja koota tähän testit halutulla tavalla ja tallentaa ne kantaan muistiin, jolloin ne lisätään automaattisesti ajattavien testien listaan. Mahdollisuuksia on useita, kuten myös toteutustapoja, mutta tarkoituksena on parantaa ja nopeuttaa testien luontia.

Testeillä voidaan myös helposti simuloida useampaa käyttää, jotka käyttävät ohjelmaa samaan aikaan. Tämä onnistuu käyttämällä useampaa Selenium-palvelinta samaan aikaan ja suorittamalla haluttuja testejä.

9 YHTEENVETO

Projektin alussa asetettiin vaatimukseksi toteuttaa automaattinen testausjärjestelmä, jolla pystytään parantamaan ja nopeuttamaan ohjelmistojentestausta. Ongelmakohtina nähtiin oikeiden ohjelmien valintaan liittyvät asiat ja testaukseen liittyvät tarpeet.

Projektin lopullisena tuotoksena saatiin Aspdia Oy:lle toteutetuksi automaattinen testausjärjestelmä. Järjestelmällä pystytään kasaamaan, asentamaan, varmuuskopioimaan ja testaamaan ohjelmia automatisoidusti, joko ajastetusti tai käyttäjän tarpeen mukaan. Järjestelmällä on mahdollista tarkkailla yrityksen tuotannossa olevien ohjelmien tilaa ja saada sähköpostivaroitus vikatilanteissa. Uudistettu järjestelmä tarjoaa ajallista säästöä ja tarkkuutta testauksen toteutukseen ja tuotantoon vienteihin. Testien päivitys ja suunnittelu ohjelmien uudistuessa vaatii edelleen oman aikansa, mutta testien päivitykseen käytettävä aika lyheni selvästi testien modulaarisuuden ansiosta. Yksittäiset ohjelman muutokset pystytään toteuttamaan testeissä helposti. Uusien testien suoritus tapahtuu automaattisesti uuden koonnin ja asennuksen yhteydessä, joten testaajan ei tarvitse tehdä koko ohjelman testausta uudelleen käsin pienten muutosten jälkeen. Kokonaisuudessaan järjestelmä tarjoaa hyvän pohjan paremmalle tavalle toteuttaa testausta, vaikkakin parannusmahdollisuuksia on tarjolla. Pelkästään testien tekemiseen tarkoitetulla käyttöliittymällä voitaisiin nopeuttaa päivityksien tekemistä huomattavasti.

VIITELUETTELO

[1] Pressman Roger S. Software Engineering Aprattitioner's Approach. 4. painos. European adaptation: 1997.

[2] Gao Jerry Zeyu, Tsao H.-S. Jacob, Wu Ye, Testing and guality assurance for component-baset software. Artech House: 2003.

[3] Fewster Mark, Graham Dorothy, Sofware Test Automation Effective use of test execution tools. Addison-Wesley: 1999.

[4] Selenium IDE Features [verkkodokumentti, viitattu 25.03.2009]. Saatavissa: http://seleniumhq.org/projects/ide/.

[5] Selenium Documentaion[verkkodokumentti, viitattu 28.04.2010]. Saatavissa: http://seleniumhq.org/docs/.

[6] LogginSelenium - Introduction [verkkodokumentti, viitattu 28.04.2010]. Saatavissa: http://loggingselenium.sourceforge.net/index.html.

[7] Xpath Tutorial [verkkodokumentti, viitattu 28.04.2010]. Saatavissa: http://www.w3schools.com/XPath/default.asp.

[8] XPather Add-ons for Firefox [verkkodokumentti, viitattu 28.04.2010]. Saatavissa: https://addons.mozilla.org/en-US/firefox/addon/1192.

Selenium IDE HTML -testitapaus lyhennetty esimerkki

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head profile="http://selenium-ide.openqa.org/profiles/test-case">
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<link rel="selenium.base" href="URL " />
<title>tilaajavastuu</title>
</head>
<body>
<table cellpadding="1" cellspacing="1" border="1">
<thead>
<tr><td rowspan="1" colspan="3">tilaajavastuu</td></tr>
</thead><tbody>
<tr>
                <td>open</td>
                <td>/tilaajavastuu.fi/sitefront/index.jsp</td>
                <td></td>
</tr>
<tr>
                <td>assertTitle</td>
                <td>Tervetuloa tilaajavastuu.fi palveluun!</td>
                <td></td>
</tr>
<tr>
                <td>clickAndWait</td>
                <td>link=Palveluhinnasto</td>
                <td></td>
</tr>
<tr>
                <td>assertTitle</td>
                <td>Palveluhinnasto</td>
                <td></td>
</tr>

...

</tbody></table>
</body>
</html>
```

Selenium IDE Java -testitapaus lyhennetty esimerkki

```
package com.example.tests;

import com.thoughtworks.selenium.*;
import java.util.regex.Pattern;

public class Untitled extends SeleneseTestCase {
	public void setUp() throws Exception {
		setUp("URL ", "*chrome");
	}
	public void testUntitled() throws Exception {
		selenium.open("/tilaajavastuu.fi/sitefront/index.jsp");
		assertEquals("Tervetuloa tilaajavastuu.fi palveluun!",
		selenium.getTitle());
		selenium.click("link=Palveluhinnasto");
		selenium.waitForPageToLoad("30000");
		assertEquals("Palveluhinnasto", selenium.getTitle());
		...
	}
}
```

Selenium testi HTML suite -tiedosto

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>

<meta content="text/html; charset=UTF-8" http-equiv="content-type" />

<title>TestSuite</title>

</head>

<body>

<table id="suiteTable" cellpadding="1" cellspacing="1" border="1"
class="selenium"><tbody>

<tr><td><b>TestSuite</b></td></tr>

<tr><td><a target="testFrame" href="NewTest.html">NewTest</a></td></tr>

</tbody></table>

</body>

</html>
```

Selenium testi tulos tiedosto

```
<html>
<head><style type='text/css'>
body, table {
   font-family: Verdana, Arial, sans-serif;
   font-size: 12;
}

…

</style><title>Test suite results</title></head>
<body>
<h1>Test suite results </h1>
<table>
<tr>
<td>result:</td>
<td>passed</td>
</tr>

…

<table>
<tbody><tr class="title status_passed"><td>Overall Tests </td></tr>
<tr class="status_passed"><td><a href="#testresult0">TEST CASE A</a></td></tr>
</tbody></table></td>
<td> </td>
</tr>
</table><table><tr>
<td><a name="testresult0">./Test.html</a><br/><div>
<table border="1" cellpadding="1" cellspacing="1">
<thead>
<tr class="title status_passed"><td rowspan="1" colspan="3">tilaajavastuu</td></tr>
</thead><tbody>
<tr class="status_done" style="cursor: pointer;">
                <td>open</td>
                <td>/tilaajavastuu.fi/</td>
                <td></td>
</tr>

…

</tbody></table>
</div></td>
<td> </td>
</tr></table><pre>
info: Starting test /selenium-server/tests/Tilaajavastuu.fiLoginSearchLogout.html
info: Executing: |open | /tilaajavastuu.fi/ |  |
info: Executing: |clickAndWait | link=Kirjaudu palveluun |  |
info: Executing: |clickAndWait | form_login:login_submit |  |

…

</pre></body></html>
```

Raportointia varten lisättävät asetukset

```
@Before
public void setUp() {
    final String resultPath = "absolute-path-to-where-your-result-will-be-written";
    final String resultHtmlFileName = resultPath + File.separator + "result.html";
    final String resultEncoding = "UTF-8"
    loggingWriter = LoggingUtils.createWriter(resultHtmlFileName, resultEncoding);

    LoggingResultsFormatter htmlFormatter =
        new HtmlResultFormatter(loggingWriter, resultEncoding);
    htmlFormatter.setScreenShotBaseUri(""); // this is for linking to the screenshots
    htmlFormatter.setAutomaticScreenshotPath(resultPath);

    // wrap HttpCommandProcessor from remote-control
    LoggingCommandProcessor myProcessor =
        new LoggingCommandProcessor(new HttpCommandProcessor(your-configs),
htmlFormatter);
    selenium = new LoggingDefaultSelenium(myProcessor);
    selenium.start();
}

@After
public void tearDown() {
    selenium.stop();
    try {
        if (null != loggingWriter) {
            loggingWriter.close();
        }
    } catch (IOException e) {
        // do nothing
    }
}
```

CruiseControl config.xml

```
<!-- start TrueDiligence -->
  <project name="TrueDiligence" buildafterfailed="false" forceOnly="true">
  <property name="projectname" value="TrueDiligence" />
    <listeners>
      <currentbuildstatuslistener file="logs/${project.name}/status.txt" />
    </listeners>
    <bootstrappers>
      <svnbootstrapper localWorkingCopy="projects/${project.name}/${projectname}" user-
name=" " password="" />
      <antbootstrapper anthome="projects/compile-config-build/compile/apache-ant-1.7.0"
buildfile="projects/${project.name}/${projectname}/build.xml" target="clean" />
    </bootstrappers>
    <modificationset quietperiod="10">
      <!-- touch any file in TrueDiligence project to trigger a build -->
      <!--<filesystem folder="projects/${project.name}" />-->
      <svn localWorkingCopy="projects/${project.name}/${projectname}" RepositoryLoca-
tion=”URL” username="cruisecontroluser" password="" />
    </modificationset>
    <schedule interval="3600">
      <ant anthome="projects/compile-config-build/compile/apache-ant-1.7.0" build-
file="config-build/build/${project.name}/service-build.xml" target="build"/>
    <!--<pause starttime="0000" endtime="2300" />
                <pause starttime="2320" endtime="2359" />
      <pause day="saturday" starttime="0000" endtime="2359" />
      <pause day="sunday" starttime="0000" endtime="2359" />-->
    </schedule>
    <!--<log dir="artifacts/${project.name}">-->
                <log>
                  <deleteartifacts every="5" unit="DAY" />
      <merge dir="projects/${project.name}/reports/junit" />
    </log>
    <publishers>
      <onsuccess>
                <artifactspublisher dest="artifacts/${project.name}"
                File="projects/${project.name}/${projectname}/dist/truediligence.war" />
      </onsuccess>
    </publishers>
  </project>
  <!-- end TrueDiligence -->
```

Projektin build.xml –tiedosto

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="build" default="build" basedir="location">

		<property file="./build.properties"/>

		<!-- Our dependencies on external libraries -->
		<fileset id="t.lib.jars" dir="${t.ant.app.dir}">
			<include name="**/*.jar"/>
		</fileset>

		<!-- Define class paths for the different ant tasks. -->
		<path id="ant-contrib.classpath">
			<fileset refid="t.lib.jars"/>
		</path>

		<taskdef	resource="net/sf/antcontrib/antcontrib.properties"	classpath-
			ref="ant-contrib.classpath" />

		<trycatch>
			<try>
			<import file=". /build.xml" />
			</try>
			<catch>
			<echo message="The file does not exist!" />
			</catch>
		</trycatch>


 <target name="build" depends="dist">
		<!--<antcall target="send"/>-->
		</target>

		<target name="send" >
			<ftp server="address"
			userid="selenium"
			password="selenium">
			<fileset  dir="./dist">
			</fileset>
			</ftp>
		</target>



</project>
```